Wireless Networking in Future Factories: Protocol Design and Evaluation Strategies

D i s s e r t a t i o n

zur Erlangung des akademischen Grades

doctor rerum naturalium (Dr. rer. nat.)
im Fach Informatik

eingereicht an der
Mathematisch-Naturwissenschaftliche Fakultät
der Humboldt-Universität zu Berlin

von
Roman Naumann, M. Sc.

Präsidentin der Humboldt-Universität zu Berlin
Prof. Dr.-Ing. Dr. Sabine Kunst

Dekan der Mathematisch-Naturwissenschaftliche Fakultät
Prof. Dr. Elmar Kulke

Gutachter/innen:
1. Prof. Dr. Björn Scheuermann
2. Prof. Dr. ir. Geert Heijenk
3. Prof. Dr.-Ing. Lars Wolf

Tag der mündlichen Prüfung: 29. November 2019

WIRELESS NETWORKING IN FUTURE FACTORIES: PROTOCOL DESIGN AND EVALUATION STRATEGIES

I declare that I have completed the thesis independently using only the aids and tools specified. I have not applied for a doctor's degree in the doctoral subject elsewhere and do not hold a corresponding doctor's degree. I have taken due note of the Faculty of Mathematics and Natural Sciences PhD Regulations, published in the Official Gazette of Humboldt-Universität zu Berlin no. 42 on July 11 2018.

Roman Naumann

Abstract

As smart factory trends gain momentum, there is a growing need for robust information transmission protocols that make available sensor information gathered by individual machines. Wireless transmission provides the required flexibility for industry adoption but poses challenges for timely and reliable information delivery in challenging industrial environments. Protocol development can be characterized by three steps: designing a system that has desired properties, implementing this design, and validating whether the implementation has the anticipated properties. This work contributes to all of these steps but focuses on design and evaluation aspects. To this end, we first identify requirements specific to our use case and derive concrete design principles that protocols should implement. We then propose mechanisms that implement these principles for different types of protocols, which retain compatibility with existing networks and hardware to varying degrees. Last, we propose evaluation techniques for possibly proprietary networking protocols, rendering, among others, the adaption of protocols to specific industrial use cases more cost efficient.

We distinguish protocols by the extent to which nodes in the network may perform extensive computation on information. When limiting such computation to information sources, we show that prioritization tailored to the use case is a powerful tool to implement robustness against challenged connectivity. This prioritization conveys an accurate preview of information from the production process, and we provide precise bounds for the quality of that preview. By moving parts of the computational work into the network, we show that reordering queues in accordance with our prioritization scheme improves fairness among machines. Finally, we show how network coding, a technique that generates new combinations of so-far-received information on the fly and fully within the network, can benefit our use case. To implement the derived principles in the domain of network coding, we improve upon existing, prioritizing network coding schemes by introducing specialized encoding and decoding mechanisms.

The last step of protocol development is evaluation. Here, we identify that it is often challenging to assess real-world protocol implementations in network simulations. We thus propose a novel architecture that allows simulating virtually any real-world protocol implementation by leveraging modern operating system properties. We demonstrate that our approach provides sufficient performance and improves the validity of evaluation results over the state of the art.

Zusammenfassung

Industrie-4.0 bringt eine wachsende Nachfrage an Netzwerkprotokollen mit sich, die es erlauben, Informationen vom Produktionsprozess einzelner Maschinen zu erfassen und verfügbar zu machen. Drahtlose Übertragung erfüllt hierbei die für industrielle Anwendungen benötigte Flexibilität, kann in herausfordernden Industrieumgebungen aber nicht immer zeitnahe und zuverlässige Übertragung gewährleisten.

Grundsätzlich kann die Entwicklung von Netzwerkprotokollen in drei wesentliche Schritte untergliedert werden: Der Entwurf eines Systems mit gewünschten Eigenschaften, die Implementierung davon sowie die Überprüfung, ob das entwickelte System tatsächlich die gewünschten Eigenschaften aufweist. Diese Arbeit trägt zu allen drei Schritten bei, mit einem Fokus auf Design- und Evaluierungsaspekte. Hierzu identifizieren wir zunächst Anforderungen für unseren industriellen Anwendungsfall und leiten daraus konkrete Entwufskriterien ab, die Protokolle erfüllen sollten. Anschließend schlagen wir Protokollmechanismen vor, die jene Entwurfskriterien für unterschiedliche Arten von Protokollen umsetzen, die in verschiedenem Maße kompatibel zu existierenden Netzwerken und existierender Hardware sind.

Hierbei unterscheiden wir Protokolle danach, welche Knoten im Netzwerk wesentliche Berechnungen vornehmen. Anhand klassischer Protokollentwürfe, in denen wesentliche Berechnungen nur an der Quelle von Informationen vorgenommen werden, zeigen wir, wie anwendungsfallspezifische Priorisierung von Netzwerkdaten dabei hilft, zuverlässige Übertragung auch unter starken Störeinflüssen zu gewährleisten. Der vorgeschlagene Priorisierungsmechanismus übermittelt eine akkurate Vorschau von Prozessinformationen, deren Fehler wir durch präzise Schranken benennen können. Ferner zeigen wir, dass die Fairness zwischen einzelnen Maschinen durch Veränderung von Warteschlangen verbessert werden kann, wobei hier ein Teil der Algorithmen von Knoten innerhalb des Netzwerks durchgeführt wird. Schlussendlich zeigen wir wie Network-Coding, eine Technik, die vollständig innerhalb des Netzwerks neue, kombinierte Daten aus bereits empfangenen Daten erzeugt, zu unserem Anwendungsfall beitragen kann. Um die vorgeschlagenen Entwurfskriterien in der Domäne von Network-Coding umzusetzen, erweitern wir existierende, priorisierte Network-Coding-Verfahren um ein spezialisiertes Kodierungs- und Dekodierungsverfahren.

Der letzte Schritt von Protokollentwicklung ist die Evaluation. Hier stellen wir fest, dass es nicht ohne weiteres möglich ist, existierende,

industrielle Protokollimplementierungen mit simulativen Evaluationstechniken zu untersuchen. Um diesen Missstand zu adressieren, schlagen wir eine Architektur vor, die es erlaubt, nahezu beliebige Protokollimplementierungen in Simulationen zu verwenden, indem Eigenschaften moderner Betriebssysteme ausgenutzt werden. Wir zeigen, dass unser vorgeschlagener Ansatz ausreichend performant für praktische Anwendungen ist und, darüber hinaus, die Validität von Evaluationsergebnissen gegenüber existierenden Ansätzen verbessert.

Acknowledgments

First, I would like to express my sincere gratitude to my advisor Björn Scheuermann for support of my Ph.D study, related research, and for his – sometimes surprising – ability to provide in-depth feedback on complex research problems that he heard about just minutes before. I could not have imagined having a better advisor and mentor. Besides my advisor, I would like to thank my two other committee reviewers, Geert Heijenk and Lars Wolf, for their constructive feedback and hospitality during my visits.

My sincere thanks also go to Stefan Dietzel with whom I wrote most publications and performed most project work. Stefan is a very talented writer and likely had the largest impact on my academic writing skills. On top of that, his positive attitude and encouragement often helped me to keep working on research problems when I was stuck.

During my time as a researcher, I had the chance to work with several talented students. In particular, I wish to thank Marie Schaeffer and Hagen Sparka. Hagen and I often discussed compression-related topics, Marie and I shared many thoughts on prioritized network coding. I enjoyed writing research articles with both of them a lot.

I also wish to thank our research assistants Laura Wartschinski and Ben Schumacher, who helped me tackle project work and to implement our algorithms on real hardware. Over the years, I shared my office with Samuel Brack and Sebastian Henningsen, who had to endure my complaining over research problems (and the noise of my mechanical keyboard). They helped me with countless issues that came up, for which I am very grateful.

I had the opportunity to discuss ideas and conference presentations with almost everyone from the group and wish to thank Daniel Cagara, Holger Döbler, Wladislaw Gusew, Sven Hager, Olga Kondrateva, Phillipp Schoppmann, Siegmar Sommer, Steffen Tschirpke, Florian Tschorsch, and Frank Winkler for their help and valuable input.

I also wish to express my gratitude to the 29 anonymous peers who reviewed the publications that this dissertation is based on. The reviewers' feedback not only helped me refine my ideas, but it largely shaped my overall understanding of the research process.

Last but not the least, I would like to thank my family: my parents, Bernd, Cordula, Eric, and Jana; my brother and sister, Carl and Marlene; as well as my partner, Christine, for supporting me spiritually throughout writing this thesis and my life in general.

Thanks, everyone!

Overlap is possible due to the nature of anonymous peer reviews.

Contents

List of Figures

List of Tables

1
Introduction

1.1 Motivation and Challenges

Industrial production accounts for 16% of Europe's gross domestic product and, according to the European commission, is a driving motor for productivity and job creation [38]. Moreover, manufactured products account for 80% of Europe's exports, the manufacturing sector employs more than 30 million people, and at least as many jobs exist in associated service sectors [38]. Today, industry nations' manufacturing industry has to face both the ongoing trend of mass-production's relocation into low-wage countries as well as a trend towards product individualization and customization [15].

Smart factory concepts are an answer to these challenges and are also known as Industry 4.0. The latter term, coined by the German government, stems from retrospectively terming the widespread industrial adoption of electricity and the digital transformation as second and third industrial revolution, respectively; Industry 4.0 comprises the following fundamental concepts [77]: fully sensorized and actor-supported production as well as its ''smart'' algorithmic exploitation with various techniques of ubiquitous computing; cyber-physical systems, in which the physical and the digital properties of production systems merge to an extent that they cannot be meaningfully differentiated any more; as well as decentralization and self-organizing capabilities of manufacturing systems to address the increasing decomposition of the classic production hierarchy. Reliable and efficient wireless communication protocols serve as an enabling factor for this vision [81].

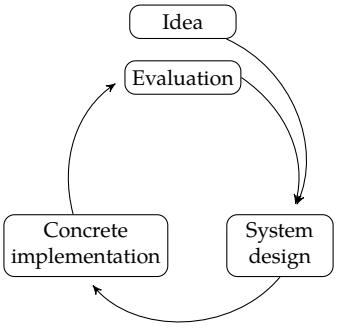
A driving motor of smart factory concepts is information from the production process itself [81]. Considered particularly useful in various sectors of manufacturing industry is sensor information: motor load sensor information from drilling machines, for instance, is used for sensor-based tool condition monitoring [35]; similarly, drilling process control is enabled by vibration sensor information [119]. Plastic industry uses material heat and pressure sensors for process control and monitoring, i. e., to automatically detect various kinds of product defects [59, 100]. Traditionally, such machines' sensor information – if available at all – was only used locally, for use in local feedback loops or to help machine operators better assess current process quality. When made available to connected computing systems, however, such information becomes more useful, e. g., for remote process control and analysis, predictive maintenance, or tracking customer complaints [55, 77].

Using wireless communication for the transmission of sensor information has benefits over traditional approaches using Ethernet or other cable-based solutions: wired connectors are not always available at all machine positions, and deploying new cables can be expensive and reduces spatial flexibility. Wireless solutions do not suffer from these limitations, rendering them an enabler for low-cost retrofitting of existing factories to benefit from smart factory concepts. On top of

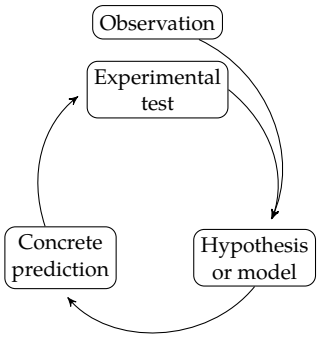
that, wireless solutions can better accommodate quickly-changing production environments and the relocation of machines, which becomes more important for individualized small-batch production.

Key challenges for efficient wireless transmission are the large quantity of available information, as well as the challenging industrial environment. Many relevant parameters are acquired as time series data with high frequency. The acquired information may over-saturate the available wireless capacity when multiple machines with several sensors each operate in parallel, as is the case in most production setups. Resulting delays may be prohibitive for industrial applications, as process deviations are detected too late. At the same time, factory floors often cover large areas and contain metal obstruction, which may impede wireless transmissions [102]. Therefore, for larger sites, single-hop communication may be insufficient to support wireless coverage throughout the factory. Especially with multi-hop communication, however, the allocation of network capacity to individual machines is challenging. It is necessary to allocate sufficient network capacity to each machine to ensure timely transmission of process information.

This work's general objective is to identify and improve upon wireless protocol design and evaluation aspects for protocols that implement transmission of process information in manufacturing-industry environments. To this end, the following chapters transition from manufacturing industry requirements to more specific aspects of protocol design and, finally, protocol evaluation. The contributions described in the individual chapters were in part implemented on hardware and tested during project demonstrations. These results characterize more readily usable solutions for specific usage scenarios. But this dissertation also contains more fundamental research results, e. g., providing asymptotic improvements or enabling previously impossible methods of protocol evaluation. These results provide benefits to a wider range of use cases, but may require more applied research to fit a specific usage scenario.



(a) Protocol development.



(b) The scientific method.

Figure 1.1: Protocols development versus the scientific method [40]. If the protocol evaluation step is negative, the design is refined and consecutive steps are repeated until the system has the desired properties. In this aspect, system development mimics the scientific method of hypothesis testing and refinement.

1.2 *Outline and Contributions*

System development can be characterized by three main steps [33, 40]: designing a system that has desired properties, implementing this design, and validating whether the implementation has the anticipated properties (compare Figure 1.1). We contribute to all three of these steps for protocol development, but focus on the design and evaluation stages. The remainder of this work is organized as follows.

Chapter 2 describes plastic industry as a tangible manufacturing-industry use case and derives principal requirements for suitable protocols, which limits the reasonable design space. An important result of Chapter 2 is that multi-hop networking capabilities are required to cover larger factory sites.

Multi-hop network protocols can be distinguished by where important computations are performed: in traditional store-and-forward architectures, such computations are performed at the sources and des-

tiations of information only, and intermediate nodes neither modify, nor act upon the forwarded information. In Chapter 3, we describe suitable protocol mechanisms for manufacturing industry that are performed at the source and destiation of information. An important idea of the presented source-based protocols is to first transform sensor information to a format in which content can be prioritized meaningfully. In particular, we contribute protocol mechanisms that leverage an energy compacting time-frequency transform to enable an early preview of sensor information. Prioritization is then used to provide that early preview even if wireless connectivity does not permit transmission of all information, e. g., due to temporary interference. We complement the source-based prioritization with a technique to obtain precise error bounds on the preview, and present a compression scheme based on combining the preview functionality with an entropy encoder.

In chapter 4, we explore how forwarding nodes can exploit knowledge of forwarded information. By retaining the store-and-forward architecture but allowing for some restricted computations on forwarded content, we strike a compromise between performance and compatibility. That is, we benefit from improved protocol performance but retain compatibility with co-existing applications on the same network and keep the requirements on nodes' computational capacities low. We show that applying prioritization based on expected information utility supports obtaining a precise approximation of process information as early as possible in larger factories. At the same time, we show that in-network prioritization supports fairness throughout the factory environment; that is, machines at different locations receive a similar amount of available network capacity.

Networking protocols that perform arbitrary computations on traffic can simplify routing decisions, improve throughput, and increases tolerance against packet loss. At the same time, they are often computationally expensive and replace the routing layer, rendering such approaches less compatible with existing protocols. A well-known group of protocols here is network coding, in which nodes generate ''combinations'' on the fly instead of forwarding traffic. As future industrial hardware is expected to provide more powerful computational capabilities, we discuss to what extent network coding can be beneficial to industrial protocol designs in Chapter 5. Thereby, we explore a fundamentally different part of the protocol design space. First, we explore to what extent key requirement from Chapter 2 and concepts from Chapters 3 and 4 translate to protocol solutions based on network coding. In particular, we identify expanding window random linear codes as promising candidate codes, which, however, suffer from inefficient decoding techniques for our use case. Consequently, we contribute a novel and use-case fitted decoding technique that reduces both computational complexity and space complexity. Further, we contribute an optimized generation process of coded packets that reduces message overhead, resulting from so-called linear dependency, and that improves the time until an approximation becomes available.

Chapter 6 concludes the larger picture of protocol development by

identifying evaluation obstacles for industrial applications – and removes major obstacles from the evaluation process. In particular, we identify the inability to easily assess network protocols in state of the art simulators if these protocols were originally implemented for test beds or real-world deployment. Thus, the state of the art often requires duplicate implementations for deployment and simulation, which may differ in subtle ways – impeding evaluation validity. This restriction is therefore both a possible cause of error and causes redundant implementation efforts. Chapter 6 contributes a novel architecture that allows running virtually any protocol implementation in discrete event simulators. We provide a detailed performance evaluation of our proposed architecture with a Linux based implementation and evaluate with existing, real-world protocols, which we also use in Chapter 4.

Protocols that are developed from scratch, with real-world deployment in mind, can also benefit from our proposed architecture, but modeling in the simulator can still help to better understand protocol mechanisms.

2
Challenges and Design Principles

2.1 Overview

In this chapter, we examine the industrial use case more closely. To this end, we focus on a tangible example process from an important manufacturing industry sector: plastic injection molding. The global market of injection moulded plastics is significant; it is expected to be worth more than 400 billion Euros by 2025 [49]. While other materials can be molded as well, in the following, we focus on plastic injection molding, which is one the most important plastics processes in use today [97].[1]

[1] In terms of production output, the European plastics industry ranks second after China and before the United States with 18% of the world production [103]. Given the inherent motivation for further automation in high wage countries, such as Europe or the United States, we consider plastics industry a good example use case.

In the following, we identify requirements on communications systems for this industrial use case, which bounds the reasonable design space. Next, we derive concrete principles that guide the proposed protocol designs and mechanisms of the following chapters. While this chapter focuses on plastic industry, the identified requirements and design principles apply to a wide range of manufacturing industry settings where parts are produced in a cyclic fashion. To make it easier to use the contributions of this work in other scenarios, we formalize the use case as an abstract system model and employ that model consistently throughout this work.

2.2 The Use Case of Injection Molding

Injection molding can be described as the process of injecting (''pushing'') molten material with high temperature and pressure into a mold (the ''form''). The mold consists of at least two parts, which are initially pressed together tightly during the material injection so that the cavity within the mold is sealed. After injection, the material cools off and hardens in the mold's cavity until, finally, the mold is opened and the solid part ejected.

In *plastic* injection molding, dried product material granulate is melted by an injection-molding machine. The material is pushed by a screw-type plunger within a barrel towards the mold with high pressure. Modern injection-molding machines expose multiple time-variant parameters from internal sensors. Such machine parameters include the position of the screw and the machine temperature and material pressure in the barrel. More recently, temperature and pressure sensors have been incorporated into the construction of the mold, which allows to measure material properties directly while the final product is formed. Pressure sensors in the mold, e. g., serve as a valuable indicator for several kinds of defective products [4, 59]. Mold temperature sensors can, e. g., be used to detect short shot defects, which occur when the mold's cavity is not fully filled with material, or variations in the cooling system resulting in improper melt temperatures [110]. Commercial sensor acquisition equipment typically queries such sensors with 100 Hz to 1000 Hz, with a total of 4 to 8 sensors covering most machines and molds [80, 104].

A key property of the injection-molding process in general is its *cyclic*

nature. The process, once set up, takes about the same duration for producing a single part each time. After producing a part, the machine cools down for a period of time. The machine exposes signals that indicate whether the mold is currently open or closed. Those signals can be used to detect whether the machine is currently injecting or cools down the mold. The ratio (penetration) of machines and molds that are sensorized, i. e., equipped with such acquisition equipment of process information, has a direct impact on required network capacity. Typical small and medium-sized enterprise (SME) [37] factory dimensions vary from 10 m² to 150 m² in diameter, with machines arranged in rows along aisles to have them accessible by, e. g., forklift trucks [105, Annex I: Industry Partner Survey] [107]. Figure 2.1 shows an example of such a factory site from a Netherlands-based SME.

Figure 2.1: An injection-molding factory.

2.3 *Information Utilization*

A number of tasks can be improved or enabled by utilizing collected sensor information from the injection-molding production process. We split these applications into two categories, based on how tolerant the applications are towards delay and inaccurate process information.

Suitable machine parameters, such as injection speed, material temperature, or holding pressure, depend on the mold used and the exact type of material injected. Imperfect machine parameters result in an increased risk of producing defective parts. Set up of a mold in an injection-molding machine is usually done based on the operator's previous experience and documentation. Especially first-time setup of molds, thus, is a lengthly trial-and-error process [120]. Even if the process is set up and eventually stable, it usually does not remain stable permanently: environmental variations in, e. g., ambient temperature, input material quality, or material dryness, can necessitate different machine parameters. Currently, detecting resulting product defects is a manual process where an operator visually inspects samples from a batch of products on a regular basis.

In addition to an unsatisfied customer, insufficient product quality can have a direct impact on the cost of production when a whole batch is returned due to defective products.

Given that sensor recordings from the production process are available for algorithmic exploitation, machine-learning algorithms can derive optimized machine parameters to improve setup time and detect defective parts as they are produced [100, 120, 130]. This promises to reduce personnel cost for quality inspection and to improve product quality. Furthermore, archiving sensor recordings of all parts produced along with identifying information, such as a part number, facilitates to track back customer complaints. Likewise, such an archive enables to retrospectively update, based on such complaints or internal findings, error detection models to avoid similar defects in the future.

As outlined in Figure 2.2, we categorize the aforementioned applications into those that are primarily delay constrained (lower-left part) and those that are more constrained by precision requirements (upper-right part): training machine learning models and information archival is a long-term endeavor, so timely delivery of sensor information is not critical. However, sensor information must have sufficient precision to

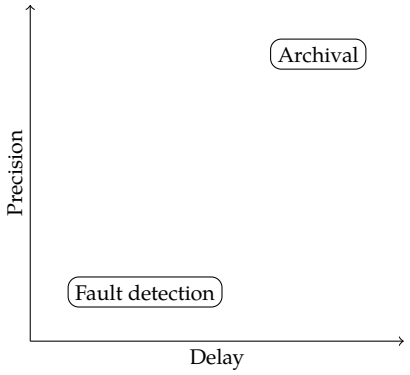


Figure 2.2: Application requirements in the industrial use case.

create detailed models. It is impossible to know at the time of system deployment what algorithmic exploitation of acquired information might be possible in the future. Considering that aspect, it is generally desirable to archive sensor information *exactly as recorded*.

Other applications of information from the production process do not necessarily require full precision of process information, but depend on timely information delivery. Most importantly, detecting defective parts must occur fast to avoid costly production of scrap material. Similarly and in a worst case scenario, strong anomalies that could not be detected in time, such as machine overheating, may cause damage to machines.

2.4 Industrial Environment Requirements

In this section, we analyze the plastic industry use case and derive major requirements and non-requirements on suitable communication technology.

Power availability: injection-molding machines generally have power outlets to supply ancillary systems with energy, so battery driven communication systems are not required. Not requiring battery-driven hardware allows for more expensive algorithmic solutions than, e. g., most wireless sensor network (WSN) algorithms [7].

Wireless technology: wired communication, however, such as Ethernet, is not usually available at machines' locations [87]. Especially older factories may not provide any communication infrastructure. As laying cables retrospectively is expensive, we expect smart factory solutions that depend on cable-based technology to suffer from lower industry adoption. Moreover, wired solutions can make relocating and rearranging machines more expensive, impeding production flexibility. Consequently, we require wireless solutions in the following.

In the EU-funded Horizon 2020 project PREVIEW [106], for instance, we encounter no Ethernet connectors near any machine at two out of four participating injection-molding SMEs. Not a single factory site had Ethernet connectors available at all machines.

Local systems: an important consideration is whether to use licensed, cellular communication systems, or often-unlicensed (e. g., industrial, scientific and medical radio bands) local area network (LAN) communication. The latter has the advantage that it allows for more control over the (often less-expensive LAN) communication hardware and does not impose running costs for cellular data. Also, a local solution offers the option to keep data that originates in the factory at a server on the factory site. Such a locality argument can help address security concerns, as factory personnel (particularly in SMEs) is often only trained to maintain physical security, such as access to the site and equipment, but not security of systems interacting over the Internet.[2] Finally, LAN solutions provide independence from cellular coverage, an important issue as factories are often located at the outskirts of cities – with suboptimal coverage. On a similar note, using LAN solutions, production is not affected by a cellular outage.

[2] In other words, personnel will usually disallow strangers from entering offices or the factory.

2.5 Design Principles for Wireless Protocols

We have established why wireless LAN communication technology is favourable for our industrial use case. We now derive and motivate more concrete requirements – which serve as guiding principles – on protocol designs on top of wireless LAN systems.

Preview functionality

Wireless communication in factory environments may suffer from severe obstruction if nodes are located at distant locations in the factory. Moreover, existing wireless technology in the factory and moving machinery (e. g., forklift trucks, gates, or cranes) can temporarily impede connectivity. Consequently, network capacity can vary significantly among machines in a factory and vary significantly over time.

To ensure continuous and reliable operation, even in case of challenged connectivity, it is important that wireless protocols resiliently deal with temporarily insufficient wireless capacity to deliver all process information. The delay-sensitive information utilization tasks that we outlined in Section 2.3 do not necessarily require full precision. Therefore, to address situations where network capacity is insufficient, protocols should implement a *preview functionality* that quickly conveys the most important information – possibly with reduced precision.

Without specific knowledge of the application scenario, it is not known how precise that preview must be to allow for, e. g., effective fault detection. Furthermore, multiple detectors may be used in parallel, each requiring a different degree of precision. To maximize flexibility, we thus require preview mechanism to allow for *incremental* updates, where the preview's precision improves over time as more network packets arrive.

Multi-hop capabilities

In larger factories, the distance between machines and the data collection system may be larger than wireless coverage permits. Deploying access points throughout the factory requires laying network cables and thereby induces additional factory-retrofitting costs. A cheaper and more flexible solution is to have each machine's communication system forward information from other machines, forming a *multi-hop* communication system.

Given typical factory dimensions, all machines can communicate with the data collection server given sufficient penetration of machines that are equipped with communication equipment. If penetration is low or the factory layout is unusual (e. g., a large empty area between groups of machines), we require additional communication systems to be deployable independent of machines. These systems do not need access to wired communication systems, as they forward information wirelessly. The additional nodes' placement is, therefore, only dependent on power availability.

Fairness

A concern of wireless multi-hop communication systems is *fairness* between machines. Commonplace networking protocols give a greater proportion of network capacity to nodes that are closer to the destination [70]. In a worst case scenario, machines that are located farthest from the data collection server may permanently suffer from insufficient network capacity to deliver process information. Avoiding such problems in retrospect, i. e., after the protocol design phase, would require more elaborate network planning, which reduces set-up flexibility and may not fully thwart against remote machines suffering from unexpected communication interruption during production. Multi-hop communication protocol designs should, therefore, implement fairness mechanisms that allocate sufficient network capacity to individual machines, regardless of their respective locations or temporarily impeded connectivity.

2.6 System Model

Our system model is an abstraction and summary of the previously described industrial use case. We describe all protocol designs in terms of this system model, which makes it easier to compare protocol mechanisms and to map mechanisms to other use cases. We assume that sensor information transmitted is acquired in form of time series data that is associated with individual production cycles, as is the case in most industrial production processes. Our network consists of u machines $m_1, \ldots, m_u$. Each machine is equipped with a number of sensors. The number of sensors can differ per machine, as can the duration of cycles and the sensors' sample rates. Therefore, we identify units of information to be transmitted as a series of sensor values, which is identified by a machine-sensor-cycle 3-tuple (i, j, k), with machine number i, sensor number j, and production cycle number k.

Typical sensor frequencies range between 100 Hz and 1000 Hz, and typical cycle durations are in the order of 1 s to 60 s, which gives between 100 and 30.000 samples per production-cycle tuple. We identify those samples as:

$$\boldsymbol{a}_{(i,j,k)} = \left(a_1, a_2, \ldots, a_{\left|\boldsymbol{a}_{(i,j,k)}\right|} \right). \tag{2.1}$$

Sensor information is recorded for a cycle duration; between such production cycles, machines have a cool down period, during which no sensor information is recorded.

We assume that, on average, link rates suffice to transmit all machines' cycle information at the rate with which they are generated. Still, bottlenecks may occur temporarily when connectivity is challenged, e. g., due to wireless interference or temporary obstruction.

Each machine i is equipped with one wireless node n_i; we call those nodes *source nodes*. One additional wireless node, termed *sink node* n_s, is installed in the network and connected to a centralized data collection server. In addition, we support an arbitrary number of *forwarding*

nodes that are not attached to a machine. These nodes extend wireless coverage for larger factories. Wireless nodes are assumed to operate non-stop, but they may fail temporarily, e. g., due to power failures, maintenance system reboots, or factory personnel relocating nodes.

The objective is to continuously transmit all available information from the machines $m_1, \ldots, m_n$ towards sink n_s. In addition to raw sensor samples, meta-data is needed to allow correct identification and processing of acquired process information. This meta-data $\text{meta}_{(i,j,k)}$ always includes the production-cycle-identifying information tuple (i, j, k) itself. Additional meta-data may include, e. g., cycle timestamps, sensor sample rates, or mold and material description fields. As the size of the meta-data can be considered small relative to the time series data size, we abstract from the application details and model meta-data as arbitrary information. Unless otherwise noted and w. l. o. g., we describe the protocol mechanisms in terms of a single production cycle (i, j, k) and thus frequently drop the index to improve readability.

Protocol mechanisms are executed repeatedly for consecutive production cycles, and in parallel for multiple machines and sensors.

2.7 *Chapter Summary*

This chapter introduced plastic injection molding, one of the most important plastic production processes in use today, as an example use case, and established how sensor information from injection molding can be used for a variety of purposes, such as automated product fault detection.

We identified several requirements on industrial systems: delivering sensor information to centralized systems should be done wirelessly to avoid expensive factory retrofitting, and with local area networks to obtain independence from cellular coverage and reduce ongoing costs. On top of being wireless, a number of design aspects must be considered in protocol and communication system designs: depending on the envisoned application, different trade-offs are feasible between sensor information precision and timely information delivery. Protocol designs for the industrial use case should utilize these trade-offs to convey important information even if connectivity is challenged. Moreover, wireless systems for process information delivery should support multi-hop communication to cover large factory sites. These multi-hop capabilities should behave fair, that is, allocate sufficient network capacity for continuous operation to individual machines regardless of their position in the factory.

Finally, we introduced the system model that we use throughout this work. The system model focuses on cyclic time-series information that can uniquely be identified by a three-tuple of machine number, sensor number, and production cycle count. Communication nodes in our system model may operate in conjunction with a machine (acting as sources), or in isolation (acting as mere forwarders).

3
Computing at the Source

3.1 Overview

This chapter is based on previous work by the author [89, © 2015 IEEE], [93, © 2017 IEEE] and parts of a collaborative work [123, © 2017 IEEE]. In particular, the evaluation of ThEMAtiC in Section 3.6 is based on work performed by co-author Hagen Sparka; initial investigations on ThEMAtiC were performed as part of a supervised study project [124].

In this chapter, we introduce networking mechanisms that are implemented at the origin of process information. That is, no requirements are imposed on nodes that forward information. Thus the mechanisms work in the same manner in a single-hop network topology (i. e., in small factories or using access points) and in larger multi-hop topologies. In particular, this chapter focuses on mechanisms that implement the preview functionality that we outlined in Section 2.5. The following chapters, in contrast, focus on the remaining design principles, which are specific to multi-hop networks.

We lay a focus on the first two contributions, since we build upon them in the next chapter. More details on the third contribution can be found in our original publication [123].

In this chapter, we describe three contributions: first, we propose a protocol mechanism that quickly conveys a characteristic representation of process information by utilizing an energy compacting time-frequency transform. Second, we show how to implement accurate and provably correct bounds for the residual error between this characteristic representation and the original process information. Last, we outline how to obtain an effective lossless compression scheme by combining an energy compacting time-frequency transform with entropy encoding techniques.

3.2 Related Work

Recent results show that production process parameters can be optimized using artificial intelligence and genetic optimization algorithms to improve product quality and minimize mold setup time [22, 120, 130]. Huang [59], in particular, demonstrates that injection molding process parameters can be optimized effectively using recordings from sensors inside the cavity. These results, hence, motivate the necessity to efficiently transmit sensor information to be used used for fault detection and parameter optimization, amongst other use cases.

The information dissemination requirements in our scenario relate to common use cases for wireless sensor networks (WSNs), which are ad-hoc networks composed of a large number of inexpensive, low-power sensor nodes. Common application scenarios include health monitoring, military, disaster recovery, and security [7, 108]. In the WSN context, a large body of literature exists on techniques that save bandwidth by means of summarization and inexpensive compression before transmission [109].

Research on compression for WSNs mostly optimizes for energy consumption and computational effort [68]. Marcelloni et al. [84, 85] propose lossless compression algorithms for WSNs. The authors exploit the high correlation between consecutive samples to reduce the range of values before using these deltas as input to an entropy encoder. Kolo et al. [69] improve upon this work by employing two entropy encoders and dynamically selecting the encoder which yields the best compression ratio. In comparison, our industrial use case is not as restrictive as WSN applications with regard to computational resources. A key difference to our work is that samples are compressed

with knowledge of preceding samples only, whereas all of our mechanisms described in this chapter make use of greater available resources and apply more advanced processing to the full information from a production cycle. This is the foundation how we achieve the early approximate previews with error bounds that we require in Chapter 2. Similar to the lossless compression scheme that we propose, Huang et al. [60] employ a discrete cosine transform (DCT) based noise filter to improve lossless compression of vibration data. Different to our work, however, the authors do not derive a domain specific noise model and refrain from using a computationally more expensive entropy-optimal encoding scheme.

Another area of research in WSNs is *lossy* compression of sensor information, surveyed by Fasolo et al. [39], in which the sink reconstructs a signal that deviates from the original sensor samples. Such lossy compression mechanisms, however, assume that a number of sensor nodes observe spatially correlated data, such as the temperature distribution within a certain geographical area. In factories, information correlation is mostly temporal, e. g., observed pressure during a manufacturing cycle, rather than spatial. Therefore, existing summarization mechanisms that aggregate information from multiple nodes are not applicable.

Christin et al. [24] and Willig et al. [136] provide an overview of industrial wireless sensor networks. The protocols and techniques described in their work are agnostic to the transmitted data. We, however, embrace the idea of an unreliable communication channel and use knowledge about the encoded sensor information for smarter retransmissions. Also, most contributions on industrial WSNs focus on energy considerations. This is especially true for industrial outdoor networks, which often run on solar power. Kulau et al. [73], e. g., describe an outdoor sensor network for smart farming applications in industrial agriculture. Gernert et al. [46] build upon this work and propose the use of mobile farming machinery as a delay tolerant networking node that collects sensor information from the field. Here, temperature and soil moisture sensors enable precise watering of parts of a field. The time frame for harvesting crops is much longer than an injection molding production cycle, so sensor information must be transmitted much faster in our scenario. In the industrial settings that we consider, source nodes are located nearby manufacturing machines, which provide the relevant sensor information, and the nodes are supplied energy via power outlets at the machines. Energy consumption is consequently much less of a concern.[1]

To implement the preview functionality required in Chapter 2, we use the discrete cosine transform (DCT). A compression algorithm very similar to our proposed protocol mechanism is the JPEG still picture compression standard [133]. After applying several transformations to linearize the two-dimensional input information, JPEG uses the DCT to prioritize more important information that characterizes the input image's main features. JPEG decompression also supports incrementally increasing image quality, not unlike our approach. Apart from

[1] Injection molding requires significant amounts of energy to constantly melt the product material and maintain a high pressure within the system. The energy consumption of wireless nodes attached to the machines is insignificant in comparison.

the different domain, our protocol mechanism has two key advantages: first, it bounds the error of the signal's preview, i. e., the characteristic representation. While JPEG compression allows a preview of the image, it cannot quantify by how much the preview differs from its original. Second, our mechanism is more than a compression algorithm that is used on top of a reliable transport. Rather, it embraces unreliable transport and allows the sink to build a characteristic representation and bound its error even if network packets are lost during transmission.

3.3 *Characteristic Representation*

Recall that this chapter is concerned with protocol mechanisms that operate at the end points of communication only. The preview functionality is therefore only implemented at the source, i. e., the production machine, and a corresponding operation is performed at the processing server. We do not assume a reliable communication channel in Chapter 2, so the basic network model here is an abstract and simple packet erasure channel between each source nodes and the sink node. This model applies to a multi-hop topology as well if nodes maintain routing tables, e. g., using link-state routing as a complementary protocol [27].[2]

[2] In a multi-hop network, the number of packet erasures, that is, lost packets, will be greater, though. Packets may be lost on any of the hops, after all.

In order to convey a preview of process information quickly, each source node performs three main tasks prior to wireless transmission, which we describe in this section: (1) transform information to the frequency domain, (2) partition the resulting coefficients into blocks suitably sized for wireless transmission, and (3) reordering those blocks in accordance with a prioritization function. After each wireless transmission, the processing server will have a set of coefficient blocks. The next section describes how the production server calculates the preview from these blocks.

Frequency transform

The transformation step uses the DCT and utilizes its *energy compaction* property for information prioritization: broadly speaking, this means that *most* of the contained sensor information can often be approximated with *few* of the DCT's low-frequency coefficients; higher frequencies then help to provide more details. In their totality, coefficients represent the complete original information.

A key motivation behind using a frequency transform such as the DCT is that we can use it to approximate the information from the full production cycle, that is, without any gaps. When we lack coefficients at the sink node, it can still approximate the full production cycle (from the beginning to the end), albeit with less precision.

Figure 3.1 shows an example of such an approximation. Here, the figure shows the in-mold temperature of an injection-molded automotive part as it is created. It can be seen that the approximation, which is based on just 5 % of the DCT coefficients, closely follows the original signal already. In fact, the approximation mostly filters out

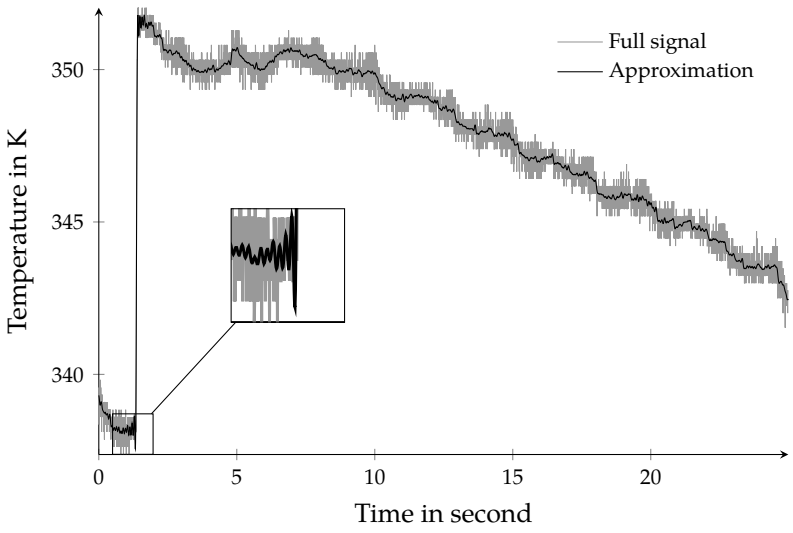


Figure 3.1: A preview of sensor information from 5 % of DCT-coefficients.

sensor noise. As the material touches the sensor, which is shown in the magnified section, the detected temperature raises rapidly. In this section of the figure, it can be seen that the approximation introduces compression artifacts near the point where the slope changes quickly. While the approximation shown may already be of sufficient quality to detect severely misconfigured process parameters, the compression artifacts motivate further updates to obtain a more precise picture of the production process state.

Formally, the DCT and its inverse, the inverse discrete cosine transform (IDCT), define mappings between a time-varying sequence of $N(= \left|\boldsymbol{a}_{(i,j,k)}\right|)$ samples $\boldsymbol{a} = (a_1, a_2, \ldots, a_N)$ and a semantically equivalent set of N cosine coefficients $\boldsymbol{X} = (X_1, X_2, \ldots, X_N)$. Here, $\boldsymbol{a}$ is the data collected at the sensor node during each production cycle, which is to be transmitted to the sink. We define the DCT and IDCT analogously to [6]:

$$\text{dct}(\boldsymbol{a}): \quad X_k = \sum_{n=1}^{N} a_n \cos\left(\frac{\pi(k-1)}{N}\left(n - \frac{1}{2}\right)\right), \tag{3.1}$$

$$\text{idct}(\boldsymbol{X}): \quad a_k = \frac{1}{2}X_1 + \sum_{n=1}^{N-1} X_{n+1} \cos\left(\frac{\pi n}{N}\left(k - \frac{1}{2}\right)\right). \tag{3.2}$$

Coefficient partitioning

The coefficient cycle $\boldsymbol{X}$ has the same size as $\boldsymbol{a}$ and is usually too large for efficient wireless transmission in a single packet [71]. We therefore partition $\boldsymbol{X}$ into blocks $\boldsymbol{B} = \left(b_1, b_2, \ldots, b_{|\boldsymbol{B}|}\right)$ that satisfy a maximum packet size constraint of P bytes. To save network capacity, our block splitting algorithm only includes relevant meta-data with the first block of each cycle, which we term *head block*.[3] The remaining blocks save space by only holding a reference to the head block and, therefore, can accommodate more coefficients.

Formally, head block b_1 for samples $\boldsymbol{X}$ consists of

$$b_1 = \left(\text{meta}, \ \ (X_1, X_2, \ldots, X_{\phi_1})\right), \tag{3.3}$$

[3] Relevant meta-data contains the machine number, sensor number and cycle number (i, j, k), but could also contain, e. g., description fields for the machine material, the mold used, and the current machine configuration.

where ϕ_1 (with $1 \leq \phi_1 \leq |\boldsymbol{X}|$) is an index chosen such that b_1 contains the maximum possible amount of coefficients while still satisfying the packet size constraint P. Consecutive blocks are built analogously, but with a reference – in the form of an (e. g., cryptographic) hash $H(\cdot)$ – to their head block:

$$b_2 = \big(H(b_1),\, (X_{\phi_1+1}, X_{\phi_1+2}, \ldots, X_{\phi_2})\big), \tag{3.4}$$

$$b_3 = \big(H(b_1),\, (X_{\phi_2+1}, X_{\phi_2+2}, \ldots, X_{\phi_3})\big), \tag{3.5}$$

$$\vdots$$

$$b_{|B|} = \big(H(b_1),\, (X_{\phi_{|\Phi|-1}+1}, X_{\phi_{|\Phi|-1}+2}, \ldots, X_{\phi_{|\Phi|}})\big). \tag{3.6}$$

Specifically, the index set $\Phi = \{\, \phi_1, \ldots, \phi_{|\Phi|} \,\}$ must satisfy

$$\forall\, \phi_i \in \Phi\colon \texttt{BinSize}(b_i) \leq P \quad \text{and} \quad \forall\, i < j\colon \phi_i < \phi_j, \tag{3.7}$$

where $\texttt{BinSize}(\cdot)$ is a coefficient block's binary, serialized size. In addition, we require that for all $i < |\Phi|$ that ϕ_i is the largest number that satisfies Equation (3.7). After transforming cycle information with the DCT and splitting sensor readings into blocks, each block is inserted into the sending queue.

Prioritization mechanism

As it is likely that the available sensor information per time unit may temporarily over-saturate available wireless capacity, an important component of our protocol is a suitable information prioritization metric. We use this metric to locally prioritize the sending queue of all wireless nodes, which may contain information from different sensors and and different production cycles. Rather than implementing a complex function that prioritizes using these individual parameters, we argue that a simple yet effective prioritization can be achieved by leveraging the DCT's *energy compaction*. Namely, we sort coefficient blocks in the sending queue such that low-frequency components are transmitted prior to high-frequency components, independent of their associated cycle id and sensor id. Due to our coefficient partitioning method, this simply means block b_i has priority over block b_j whenever $i < j$, even if the cycle-identifying tuple is different for the two blocks.

As a result, our wireless nodes first transmit information that enables the sink to approximate with similar precision all cycles that are currently transmitted. If the wireless capacity is not saturated, it is likely that the current production cycle's data is completely transmitted before the next cycle completes and is added to the transmission queue. If the channel is temporarily over-saturated, the next cycle's low-frequency coefficients will be added to the sending queue with higher priority than the current cycle's lower-frequency coefficients. Thereby, we ensure that all cycles can be approximated with low delay, and detailed representations are only transmitted when the current network capacity is sufficient.

The prioritization mechanism can be implemented efficiently using a priority queue of coefficient blocks, which can be implemented as a

heap. This way, taking the highest priority coefficient block from the priority queue only requires constant time. Insertions take logarithmic time with respect to the size of the queue, which is sufficient for our use case where queues are expected to be temporary (see Section 2.6).

3.4 *Preview Calculation*

Whenever the sink receives a block of coefficients, it performs two steps: first, it derives a current *preview* of the sensor information and then re-calculate an *error bound* on that preview.

Preview calculation

The preview calculation works as follows: let $I = \{1, \ldots, N\}$ be the set of all coefficient indices and $R \subseteq I$ the set of indices of those coefficients that are currently known to the sink. For constructing the preview $\widehat{\boldsymbol{a}}^{(R)}$ based on this knowledge,[4] the sink sets all unknown coefficients to zero and then calculates the IDCT of these coefficients.

[4] In this work, we frequently use double-index notation. To help distinguish an index from exponentiation, we add parentheses to the upper index.

That is, each coefficient with an index in R is used, and each unknown coefficient (not part of R) is assumed to be zero:

$$\widehat{\boldsymbol{a}}^{(R)} = \operatorname{idct}\left(\widehat{\boldsymbol{X}}^{(R)}\right), \quad \text{where } \widehat{\boldsymbol{X}}^{(R)} = \left(\widehat{X}_1^{(R)}, \ldots, \widehat{X}_N^{(R)}\right) \tag{3.8}$$

$$\text{with } \widehat{X}_k^{(R)} = \begin{cases} X_k & \text{if } k \in R, \\ 0 & \text{otherwise.} \end{cases}$$

When all coefficients have arrived, i. e., when $R = I$, the approximation $\widehat{\boldsymbol{a}}^{(R)}$ equals $\operatorname{idct}\left(\operatorname{dct}(\boldsymbol{a})\right) = \boldsymbol{a}$. That is, the approximation equals the original process information. Note that no additional transmissions are required for the preview functionality.

Error bounds

The provided preview's error decreases as more packets arrive at the sink. Full precision is achieved when all coefficients have arrived. Before that point, it is desirable that the sink can characterize the deviation between the current preview $\widehat{\boldsymbol{a}}^{(R)}$ and the not-yet-known signal $\boldsymbol{a}$ that was recorded by the machine. Therefore, we implement a mechanism that provides an upper bound for this error.

We define the maximum relative preview error, hereafter simply called *error*, of a preview $\widehat{\boldsymbol{a}}^{(R)} = (\widehat{a}_1^{(R)}, \ldots, \widehat{a}_N^{(R)})$ as follows:

$$\operatorname{err}(\boldsymbol{a}, \widehat{\boldsymbol{a}}^{(R)}) = \max_{k=1}^{N} \left| \frac{a_k - \widehat{a}_k^{(R)}}{a_k} \right|. \tag{3.9}$$

With this definition, a preview sample $\widehat{a}_k^{(R)}$ with $\operatorname{err}(\boldsymbol{a}, \widehat{\boldsymbol{a}}^R) = p$ implies that the true value, a_k, is in the range

$$a_k \in \left[\frac{\widehat{a}_k^{(R)}}{1+p}, \frac{\widehat{a}_k^{(R)}}{1-p} \right]. \tag{3.10}$$

To estimate the current error at the sink, we introduce a concept that we term *e*-values. Intuitively, the *e*-value $e(\boldsymbol{a}, M)$ characterizes the error if coefficients with indices in $M \subset I$ are still *missing*, i. e., when $R = I \setminus M$. More formally,

$$e(\boldsymbol{a}, M) = \operatorname{err}\Big(\boldsymbol{a}, \operatorname{idct}\big(\widehat{\boldsymbol{X}}^{(I \setminus M)}\big)\Big). \tag{3.11}$$

Because *e*-value calculation requires knowledge of $\boldsymbol{a}$, *e*-values can only be calculated by source nodes. Our goal is to allow the sink node to obtain knowledge about the *e*-value for the currently missing coefficients. Taking into account lost packets due to unreliable transmissions, the sink may miss any subset of the original coefficients. Transmitting *e*-values for all possible combinations of missing coefficients would imply exponential overhead and is thus not viable. We therefore transmit only specific *e*-values, selecting them in a way that still allows for accurate error bounds.

Assume the sensor node sends a packet containing a block of coefficients $X_j, \ldots, X_k$. First, we include an *e*-value e_{rest} that characterizes the remaining error, assuming the sink has received all coefficients $X_1, \ldots, X_k$ and none of the coefficients $X_{k+1}, \ldots, X_N$:

$$e_{\text{rest}} = e\big(\boldsymbol{a}, \{\, k+1, \ldots, N \,\}\big). \tag{3.12}$$

In addition, we observe that transmission errors are likely to affect single packets. Consequently, we include the *e*-values associated with missing *only* the predecessor block or the successor block and no other coefficients; we call these e_{pred} and e_{succ}, respectively.

Using these three *e*-values, the sink can calculate error bounds for any transmission errors that affect at most two consecutive coefficient blocks. To this end, we use the following bound, which we formally prove in Section 3.A.

Theorem 1 (Cumulative bound)**.** *Let K and L be sets of missing coefficient indices. Then*

$$e(\boldsymbol{a}, K) + e(\boldsymbol{a}, L) \geq e(\boldsymbol{a}, K \cup L). \tag{3.13}$$

That is, the sum of *e*-values for sets of missing coefficients yields an upper bound on the actual *e*-value for the sets' union. Applying this theorem, our protocol estimates the current error at the sink by adding known *e*-values associated with missing coefficients.

Consider the following example: let a set of sensor values be fully described by coefficients $X_1, \ldots, X_{10}$, and let each block only contain a single coefficient. The sensor node has transmitted the first seven coefficients, but the third coefficient's block was lost during transmission. The sink, hence, has knowledge of:

$$X_1, X_2, \quad X_4, X_5, X_6, X_7.$$

As $e\big(\boldsymbol{a}, \{\, 3 \,\}\big)$ was transmitted as e_{pred} in the packet carrying X_4, as well as as e_{succ} in X_2's packet, the sink calculates:

$$E = e\big(\boldsymbol{a}, \{\, 3 \,\}\big) + e\big(\boldsymbol{a}, \{\, 8, 9, 10 \,\}\big), \tag{3.14}$$

which is equal to or larger than $e(\boldsymbol{a}, \{3, 8, 9, 10\})$ by the above theorem, and thus again provides a valid upper bound on the error. When no packets are missing, the error bound provided by e-values equals the actual error as calculated at the source. The fewer packets are missing the closer the bound is to the error at the source, since Theorem 1 has to be invoked less often.

If too many e-values are missing to apply Theorem 1, the last-known preview can still be used with its error bound while waiting for the re-transmission of missing coefficients and e-values.

Smart retransmissions

So far, we have implemented a mechanism that transcodes time-varying sensor information using the DCT to prioritize characteristic information during transmission, as well as a mechanism to bound its errors. While a key feature of our transmission scheme is its resilience to packet losses, it is desirable to retransmit missing coefficients to achieve eventual transmission of complete, precise information. Moreover, important low-frequency coefficient blocks may be lost during transmission, making retransmissions a necessity.

Different to sequential protocols, such as the transmission control protocol (TCP), we can use the sending queue's priority for retransmissions as well. To implement retransmissions, the source node re-uses the original transmission queue of frequency components that have not yet been sent; to trigger retransmissions, the processing server periodically sends acknowledgment vectors for each production cycle, with one bit signaling reception of one coefficient block. In addition, each node sets a timeout when sending a block. Having received a positive acknowledgment for a block, the sensor node marks that block as successfully transmitted. As soon as two negative acknowledgments or two timeouts are received for a block (or any combination thereof), the sensor node re-inserts that block to its transmission queue.[5] Since the queue is ordered by increasing coefficient frequency, the retransmission mechanism ensures that the sink receives blocks with important coefficients first.

[5] Requiring two timeouts or negative acknowledgments avoids spurious retransmissions that can otherwise result from media access control (MAC) layer queues.

3.5 *Efficient Preview Without Incremental Updates*

The mechanisms discussed so far provide fine grained, incremental updates to the approximation at the sink. We consider these updates to process information important for use cases where several product fault detection algorithms may require different degrees of precision. Also, by not making any assumptions on how precise the preview has to be, these approaches' are applicable to a wide range of use cases.

When designing a system for a specific use-case, however, it is plausible that the required preview precision for timely fault detection is known beforehand and only a single precision level is required. In this special case, it is still desirable to eventually obtain all process information with maximum precision for archival-category use cases, but incremental updates provide no benefit. Although such scenarios are not the focus of this work (we require incremental updates in Section 2.5), this section outlines how compression can be improved in

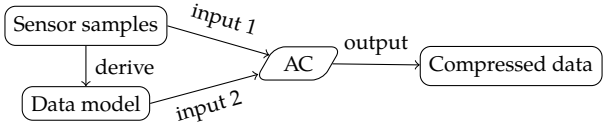


Figure 3.2: Compression overview. © 2017 IEEE

these scenarios and the amount of information to be transmitted over the network thereby reduced.

The proposed compression algorithm, named Two stEp Model AC-Compressor (ThEMAtiC), works by combining a DCT-based approximation, similar to Section 3.3, with an *entropy encoder* that provides a lossless compression of the *residual error* between the approximation and the full signal. As soon as the compressed samples are fully received by the sink, the approximation can be re-calculated to fully reflect the original signal.

Effective lossless compression with entropy encoding

Huffman codes add overhead if symbol probabilities do not follow an exact binary pattern.

Our compression algorithm is based on arithmetic coding (AC) [137], a form of lossless entropy encoding. Unlike Huffman codes, AC does not impose restrictions on symbol probabilities for achieving near-optimal compression, which usually results in an improved performance [112]. Arithmetic coding requires a data model, i. e. symbol probabilities, as a parameter. The more precise the model, the better the compression that AC achieves [45]. Figure 3.2 summarizes our AC-based compression: sensor samples and a data model are used as input to an arithmetic coder. Our main contribution is the derivation process for a data model that enables highly effective compression for typical industrial sensor information streams.

We derive our data model in two steps, which are shown in Figure 3.3: first, a DCT is applied to the original sensor information. Exploiting that the DCT results in a number of coefficients that represent information of the original signal with decreasing precision, we select a subset of the DCT's output based on energy (not frequency)

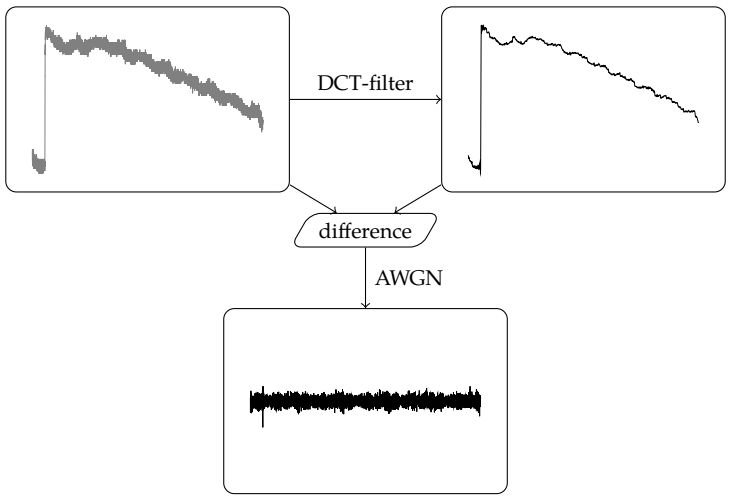


Figure 3.3: Data model derivation. © 2017 IEEE

to obtain a lossy compression mechanism. Using the DCT in this way is similar but not identical to applying a low pass filter to the original signal: it approximates its basic shape, but filters noise components. The resulting overall compression algorithms, however, is lossless, as we compress the remaining error next.

Second, we model derivations – which represent sensor noise and approximation error – from this approximation using an additive white Gaussian noise (AWGN) model. The resulting data model consists of two components: (a) a subset of all DCT coefficients and their respective frequencies and (b) the AWGN variance. These two components together give us the probability that a sample at a given point in time assumes a certain value in the value range. Note that the way in which we choose the cosine coefficients here is especially well suited to separate the AWGN from the signal. In the frequency domain, the former will contribute uniformly to the energies of all frequencies, while the signal will produce a limited number of high energy coefficients.

Using AWGN to jointly model approximation error and sensor noise strikes an effective balance between complete customization of the probability distribution and using a generic distribution for all transmissions. The former would cause prohibitive overhead, as the whole table would need to be transmitted, whereas the latter would sacrifice compression performance by leaving domain knowledge unused.

Finally, we use the per-sample probability distributions and the unmodified sensor information as input to arithmetic coding. Both the resulting compressed sensor information and the data model are then sent over the wireless network to the receiver node. Using the received data model, this node reconstructs the per-sample probability distribution and applies arithmetic coding decompression to the compressed data.

3.6 Evaluation

To evaluate the proposed mechanisms, we use real-world sensor readings that were obtained during injection molding production processes. Our evaluation focuses on three main aspects:

1. characterization of the residual error using the proposed DCT-based pre-processing and prioritization technique;
2. comparing the actual approximation error to the error bound; and
3. the achieved compression ratio of the proposed algorithm.

Unless otherwise noted, data points in plots show the arithmetic mean, and error bars indicate 95% confidence intervals. Error bars might not always be visible in the figures when the error is very small.

Methodology

Evaluation data The real-world dataset used in this evaluation is heterogeneous and contains a variety of semantic subsets. The dataset

was recorded during injection molding of an automotive part, where the holding pressure was modified in a controlled way. Therefore the dataset not only contains recordings from ''good'' parts, but also includes recordings of parts with both aesthetic and technical issues. By varying the holding pressure, various defects such as short shots, sink marks, scratch marks, and overpacking were induced. Overall, our dataset is a collection of sensor recordings for more than 160 produced plastic parts, each containing the readings of five different sensors, measuring four different physical quantities with different noise characteristics and curve shapes. Here, we do not discern between the over 20 unique subsets forming our real-world dataset. Instead, we deem the presence of faulty parts an important property of our evaluation data, as the timely transmission and effective compression of faulty parts' data is at least as important as compression of good parts.

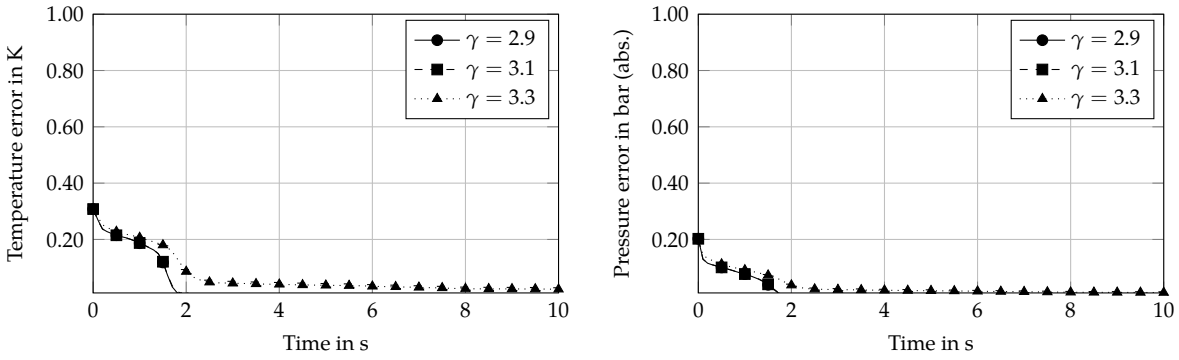
Each production cycle was recorded by five sensors at 500 Hz sample rate each. The part produced is rather large with a 25 s cycle duration; thus, a total of 12 500 samples per sensor per cycle was collected. Two of the sensors were located at the injection molding machine and track (1) the position of the screw that injects plastic into the mold and (2) pressure in the machine. The remaining three sensors are located in the mold's cavity and track (3) cavity pressure and cavity temperatures in the (4) front and (5) back. Temperature at more than one position can, for instance, be used to detect non-fills, that is, faults that stem from insufficient fill in the mold. Temperature and pressure information use Kelvin and bar (absolute), respectively, as units of measurement.

Note that we use this real-world dataset in the evaluation of Chapter 4 and Chapter 5 as well.

Simulation set-up To evaluate transmission time, representation error, and our error bound's quality, we use the discrete-event network simulator ns-3 [56] (version 3.29). A single 30 m hop, from machine to sink, is simulated via YANS Wifi model [76] with IEEE 802.11g MAC and 2.4 GHz physical layer. We assume a blocked line of sight between machine and sink. The effects of multipath propagation and large-scale path loss are accounted for by the Rayleigh and log-distance propagation loss models, one superimposed on the other [54]. The source sends at a constant rate of 1 Mbit/s; in real scenarios, a suitable data rate can be determined experimentally or by estimation. The acknowledgment interval is 200 ms, which we found to be a good compromise in most scenarios. We set the maximum packet size constraint P (see Section 3.3) to 1 kB,[6] which helps to reduce packet loss [71].

[6] Thus, $\approx$ 240 coefficients plus meta data are stored in each packet.

We use all 165 pre-recorded production cycles for transmitted production cycles in the simulation. Each measurement is performed for 600 s simulated time after a 100 s transient phase. The simulated source has four sensors and sends 8 production cycles before it stops. Simulations are executed 40 times; each repetition uses a separate sub-stream of ns-3's MRG32k3a pseudo-random number generator to ensure uncorrelated results [75]. Pseudo-randomness is used in the simulation's wireless fading model, the ns-3 bit error model, which is affected by



(a) Temperature error.

(b) Pressure error.

Figure 3.5: *Average* transmission error.

fading and path loss, and exponentially distributed variations in packet send times, which we use to avoid collisions and other synchronization effects.

Compression set-up Since ThEMAtiC compresses agnostic of underlying protocol mechanisms, we restrict the comparison to other compression algorithms. We compare the compression ratio to the WSN-specific compression algorithm SC [84], which is based on compressing information deltas, making it well-suited for compressing sensor information. In addition, we compare our algorithm against three computationally more expensive general-purpose compression algorithms: the Burrows-Wheeler-transformation-based [16] `bzip2`, the newer `xz` compression, which, among other techniques, also uses arithmetic coding, and the recently standardized [9] `brotli` algorithm. For evaluation, we used the highest, i. e., most effective, compression level that each algorithm supports. The fraction of coefficients used in the approximation step for each sensor is given in Table 3.1.

Table 3.1: Number of sensor coefficients. © 2017 IEEE

Sensor	coefficients
Screw position	2.1 %
Machine pressure	4.5 %
Cavity pressure	5.0 %
Cavity temperature front	3.5 %
Cavity temperature back	3.0 %

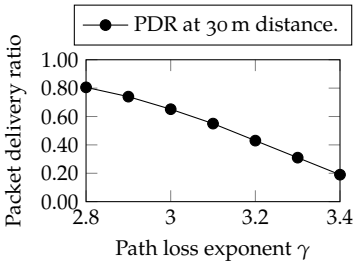
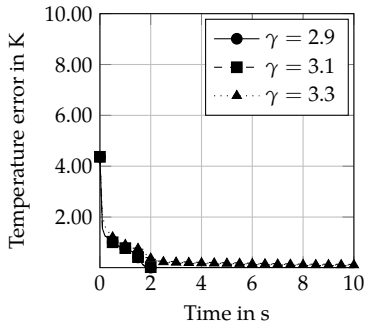


Figure 3.4: Packet delivery ratio (PDR) for varying path loss exponents. Without retransmissions, i. e., using broadcast MAC frames.

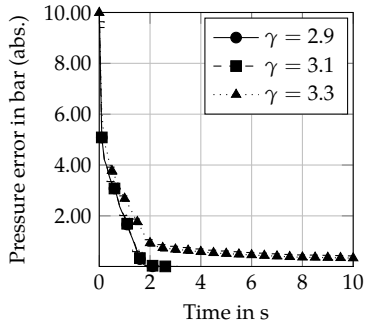
Approximation quality

To assess the quality of the DCT-provided approximation, we consider both the *average* error over all samples, and the *maximum* error between two samples of the approximation and the full process information. The average error serves as an indicator for general approximation quality, whereas the maximum error is a better fit to detect compression artifacts such as the artifact shown in Figure 3.1. We vary the log-distance exponent γ in our simulation to match different reception conditions; Figure 3.4 provides a convenient mapping of path loss exponent to packet loss in our simulated scenario.

Figure 3.5 shows the average absolute error of the approximation, as soon as it is available, compared to the full signal. Approximation error of the in-mold temperature sensor is shown in Figure 3.5a whereas Figure 3.5b gives the pressure error of the pressure error. As can be seen, the error reduces quickly when the most important coefficients have arrived. In particular, $\gamma = 2.9$ and $\gamma = 3.1$ result in the approximation's error quickly converging to zero. It took less than 1.6 s until



(a) Temperature error.



(b) Pressure error.

Figure 3.6: *Maximum* transmission error.

the error of the preview drops below 100 mK for $\gamma \leq 3.1$ and 0.6 s for an preview with less than 100 mbar residual error. Even with $\gamma \geq 3.3$, at which we observe packet loss rates of 70% and more, the error drops below 100 mK and 100 mbar within 2 s and 0.8 s, respectively, which is much shorter than the average production cycle duration. Notably, the average temperature error drops slower, as it is more affected by noise and thus requires more coefficients to approximate all details.

Figure 3.6 shows the identical scenario's results, but gives the *maximum* approximation error over time, again averaged over all runs and applicable sensors (two pressure sensors, two temperature sensors). As expected, the observed error is higher due to compression artifacts, but it still quickly conveys precise process information. While the temperature sensor exhibits a higher level of noise, the pressure changes more rapidly, making pressure harder to approximate without artifacts. This effect can be seen by comparing Figure 3.6 to Figure 3.5: in Figure 3.6, the temperature sensor's preview gains precision faster, whereas Figure 3.5 shows the opposite relationship.

Error-bound closeness

An important feature of our approach is the preview functionality's error bound. Therefore, we compare our transmission scheme's error bound – which can be calculated at the sink – with the actual representation error. Figure 3.7 shows the maximum error ($\gamma = 3.3$) from Figure 3.6a and, in addition, gives the error bound as calclated by the sink. Here, the error on the y-axis is given as maximum relative error, in accordance with our error metric Equation (3.9) from Section 3.4.

We observe that the mean difference between error bound and actual error during transmission is only 0.2 base points (e. g., hundredths of a percentage point); and even the maximum difference per cycle (averaged over all production cycles) is merely 6.5 base points.[7]

Summarizing, our proposed mechanism fulfills the preview require-

[7] In part, these low errors can be attriuted to the already-low relative error of the sensor.

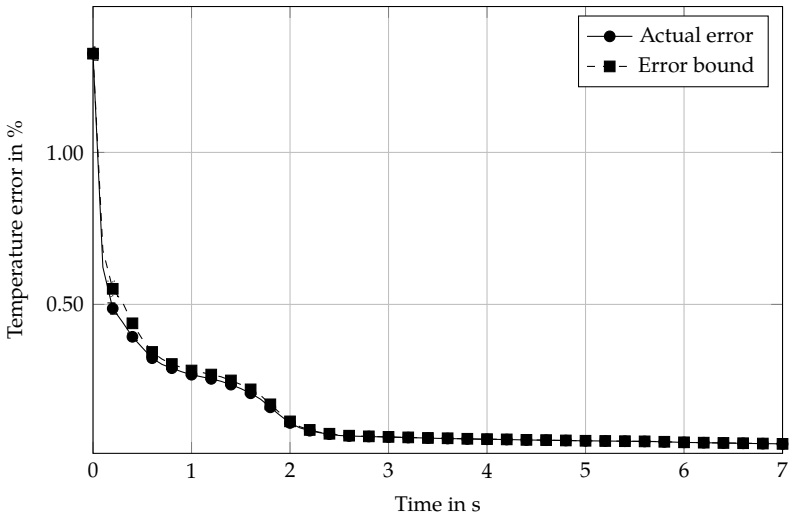


Figure 3.7: Error bound vs actual maximum error.

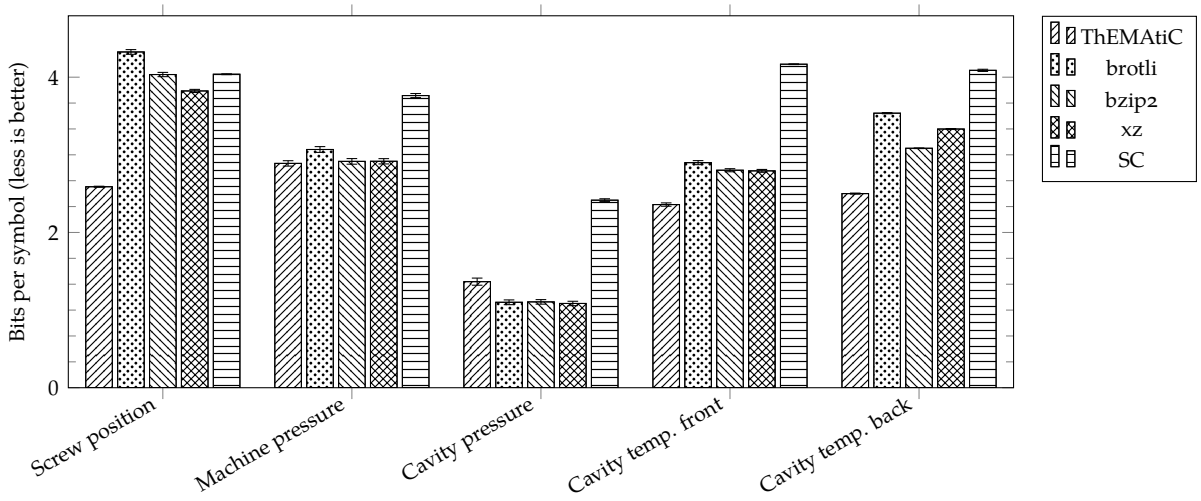


Figure 3.8: Comparison of compression results. © 2017 IEEE

ment we identified in Section 2.5. Due to the DCT, the sensor information representation's error bound improves quickly with increasing transmission time, allowing for quick responses based on characteristic features. Finally, the transmitted error bound closely resembles the actual error to the point that it is almost identical, and thereby enables informed decision making at the sink.

Compression ratio

Last, Figure 3.8 compares the compression effectiveness of ThEMAtiC to `brotli`, `bzip2`, `xz`, and SC. The x-axis groups by sensor type, the y-axis shows compressed total size in bits divided by the number of samples N; thus, a low number of bits per sample indicates a high compression ratio. In general, the energy-consumption-focused WSN algorithm SC is the least effective for all sensor types when compared to the computationally more complex general purpose algorithms, so we focus on comparing ThEMAtiC to those general purpose algorithms.

Our dataset's cavity temperature readings exhibited the highest degree of noise, so the good compression of ThEMAtiC matches our expectations: ThEMAtiC compresses cavity temperature between 16% and 29% better than the three general purpose algorithms.

For the cavity pressure sensor, the DCT requires a large number of coefficients to approximate the sensor readings with sufficient precision (see also Table 3.1), resulting in performance comparable to but not surpassing that of generic compression mechanisms. For machine pressure, the achieved compression of ThEMAtiC is better than `brotli`. The difference in performance to the remaining two general purpose compression algorithms is statistically insignificant (for $p = 0.05$). Summarizing, ThEMAtiC provides superior or on-par performance for four out of five machine and in-cavity sensors, rendering it a good fit for industrial applications with characteristings similar to our use case.

3.7 Chapter Summary

In this chapter, we introduced networking and compression techniques that are implemented at the sources of process information. The contributions described in this chapter have in common that they first transform the sensor information from the production process into the frequency domain. Here, they exploit the energy compaction property of the DCT to obtain an approximation of sensor information. The first mechanism that we introduced distributes the DCT coefficients into network packets and prioritizes transmissions so that important information is delivered first. As a consequence, the processing server can quickly calculate a preview of process information that over time improves in precision. We also showed how a bound on the error of this approximation can be calculated by the server and shown that it closely follows the actual error, enabling the incremental activation of fault detection algorithms that require different degrees of precision. Last, we showed how, instead of improving the approximation incrementally, it can be utilized to construct an effective lossless compression algorithm based on entropy encoding techniques. This algorithm is useful in scenarios where incremental updates to the initial approximation are not required and it does not require custom network protocols, so it can be integrated easily with existing networking solutions.

Chapter Appendices

3.A Proof of Theorem 1

We first state and prove a lemma based on transformation matrices that we subsequently use to prove Theorem 1 from Chapter 3.

In the following, an orthogonal transformation matrix $\boldsymbol{A}$ of size $N \times N$ is used for the discrete cosine transform (DCT) and the inverse discrete cosine transform (IDCT). [8] We assume $\boldsymbol{X} = \boldsymbol{A}\boldsymbol{a}$ corresponds to the DCT and $\boldsymbol{a} = \boldsymbol{A}^T\boldsymbol{X}$ to the IDCT. Here, $\boldsymbol{a}$ and $\boldsymbol{X}$ are *matrices* of size $N \times 1$, that is, column vectors.

[8] Such a transformation matrix can be derived by scaling the DCT with a factor of $\sqrt{2/n}$ or $\sqrt{2/(n-1)}$, depending on the variant of the DCT.

Lemma 1. *For a signal $\boldsymbol{a}$ with N samples and coefficients $\boldsymbol{X} = \boldsymbol{A}\boldsymbol{a}$, let R, M, $\widehat{\boldsymbol{X}}^{(R)}$ and $\widehat{\boldsymbol{a}}^{(R)}$ be defined as in Section 3.4, the following equation holds:*

$$\left|\frac{a_m - \widehat{a}_m^{(R)}}{a_m}\right| = \left|\frac{1}{a_m}\sum_{i\in M} A_{m\,i}^T X_i\right| \quad \forall m \in \{\,1,\ldots,N\,\}\,. \tag{3.15}$$

The important property of Lemma 1 is that it allows us to describe a preview's error solely in terms of *missing* coefficients M, whereas the definition of the preview – and thus the error – is based on *received* coefficients R. This property can be seen by the use of R on the left-hand side of the lemma and M on the right-hand side.

Proof. According to its definition, the preview $\widehat{\boldsymbol{a}}^{(R)}$ equals:

$$\widehat{\boldsymbol{a}}^{(R)} = \boldsymbol{A}^T\widehat{\boldsymbol{X}}^{(R)} = \boldsymbol{A}^{(T)}(\boldsymbol{I}_N\widehat{\boldsymbol{X}}^{(R)}), \tag{3.16}$$

where $\boldsymbol{I}_N$ is the $N \times N$ identity matrix. We split the identity matrix into N sparse matrices $\boldsymbol{Z}^{(1)}, \boldsymbol{Z}^{(2)}, \ldots, \boldsymbol{Z}^{(N)}$ of size $N \times N$, their elements defined by

$$Z_{i\,j}^{(k)} = \begin{cases} 1 & \text{if } i = j = k, \\ 0 & \text{otherwise.} \end{cases} \tag{3.17}$$

Since $\boldsymbol{I}_N = \sum_{k=1}^{N} \boldsymbol{Z}^{(k)}$, we can substitute:

$$\begin{aligned} \boldsymbol{A}^T(\boldsymbol{I}_N\widehat{\boldsymbol{X}}^{(R)}) &= \boldsymbol{A}^T(\boldsymbol{Z}^{(1)}\widehat{\boldsymbol{X}}^{(R)} + \boldsymbol{Z}^{(2)}\widehat{\boldsymbol{X}}^{(R)} + \cdots + \boldsymbol{Z}^{(N)}\widehat{\boldsymbol{X}}^{(R)}) && (3.18)\\ &= \boldsymbol{A}^T(\boldsymbol{Z}^{(1)}\widehat{\boldsymbol{X}}^{(R)}) + \boldsymbol{A}^T(\boldsymbol{Z}^{(2)}\widehat{\boldsymbol{X}}^{(R)}) + \cdots + \boldsymbol{A}^T(\boldsymbol{Z}^{(N)}\widehat{\boldsymbol{X}}^{(R)}). && (3.19) \end{aligned}$$

Considering that

$$\boldsymbol{a} = \boldsymbol{A}^T(\boldsymbol{Z}^{(1)}\boldsymbol{X}) + \boldsymbol{A}^T(\boldsymbol{Z}^{(2)}\boldsymbol{X}) + \cdots + \boldsymbol{A}^T(\boldsymbol{Z}^{(N)}\boldsymbol{X}) \tag{3.20}$$

and that $\boldsymbol{Z}^{(i)}\widehat{\boldsymbol{X}}^{(R)}$ only contains zeroes if $i \in M$, but $\boldsymbol{Z}^{(i)}\widehat{\boldsymbol{X}}^{(R)} = \boldsymbol{Z}^{(i)}\boldsymbol{X}$

if $i \notin M$, we can instead write:

$$\widehat{\boldsymbol{a}}^{(R)} = \boldsymbol{a} - \boldsymbol{A}^T \left(\boldsymbol{Z}^{(i_1)}\boldsymbol{X}\right) - \left(\boldsymbol{Z}^{(i_2)}\boldsymbol{X}\right) - \cdots - \left(\boldsymbol{Z}^{(i_{|M|})}\boldsymbol{X}\right) \tag{3.21}$$

$$= \boldsymbol{a} - \sum_{i\in M} \boldsymbol{A}^T \left(\boldsymbol{Z}^{(i)}\boldsymbol{X}\right) = \boldsymbol{a} - \sum_{i\in M} \begin{vmatrix} A_{1\,i}^T X_i \\ \vdots \\ A_{N\,i}^T X_i \end{vmatrix} \tag{3.22}$$

Applying index m ($\forall m \in \{1, \cdots, N\}$) yields

$$\widehat{a}_m^{(R)} = a_m - \sum_{i\in M} A_{m\,i}^T X_i, \tag{3.23}$$

which can be used at the left-hand side of the lemma. □

Theorem 1 (Cumulative bound)**.** *Let K and L be sets of missing coefficient indices. Then*

$$e(\boldsymbol{a}, K) + e(\boldsymbol{a}, L) \geq e(\boldsymbol{a}, K \cup L). \tag{3.13}$$

Proof. Using Lemma 1, $e(\boldsymbol{a}, K) + e(\boldsymbol{a}, L)$ equals:

$$\max_{m=1}^{N} \left| \frac{1}{a_m} \sum_{k\in K} \boldsymbol{A}_{m\,k}^T \cdot X_k \right| + \max_{m=1}^{N} \left| \frac{1}{a_m} \sum_{l\in L} A_{m\,l}^T \cdot X_l \right| \tag{3.24}$$

$$\geq \max_{m=1}^{N} \left(\left| \frac{1}{a_m} \sum_{k\in K} A_{m\,k}^T \cdot X_k \right| + \left| \frac{1}{a_m} \sum_{l\in L} A_{m\,l}^T \cdot X_l \right| \right) \tag{3.25}$$

$$\geq \max_{m=1}^{N} \left| \frac{1}{a_m} \sum_{k\in K} A_{m\,k}^T \cdot X_k + \frac{1}{a_m} \sum_{l\in L} A_{m\,l}^T \cdot X_l \right| \tag{3.26}$$

$$\geq \max_{m=1}^{N} \left| \frac{1}{a_m} \sum_{i\in K\cup L} A_{m\,i}^T \cdot X_i \right| = e(\boldsymbol{a}, K \cup L). \tag{3.27}$$

□

The triangle inequality $|x| + |y| \geq |x + y|$ is used on every term between (3.25) and (3.26).

4
Multi-Hop Networks and In-Network Computation

4.1 Overview

This chapter is a revised and extended version of the author's publication [93, © 2017 IEEE]. The extended evaluation uses the evaluation techniques described in the author's publications [91, 92]. Parts of the real-world implementation in Section 4.5 were developed by co-authors Stefan Dietzel, Laura Wartschinski, and Ben Schumacher.

The trend towards smart factories necessitates wireless transmission of production process information. Protocols must transfer process information in a timely manner even with temporary wireless interference on the factory floor and regardless of machines' location in the network topology.

In this chapter, we complement the source-based mechanisms from the previous chapter with multi-hop capabilities to cover larger factory floors. This chapter introduces TANDEM, a topology-independent, wireless multi-hop network protocol that implements in-network prioritization to provide an early approximation of process information. TANDEM applies a time-frequency transformation to prioritize more valuable information, decouples multi-hop communication steps to improve robustness, and uses an acknowledgment overhearing mechanism to save transmissions. We analytically discuss reliability and fairness aspects and show, using simulations, that TANDEM's prioritization is beneficial to obtain an early preview of sensor information even in large factories. In addition and as a proof of concept, we implement TANDEM on industrial grade hardware and provide a performance overview.

4.2 Related Work

TANDEM's use case closely relates to wireless mesh networks, for which a number of routing protocols have been proposed in the last decades [8]. Ad-hoc on-demand distance vector (AODV) [101] and open link state routing (OLSR) [28] are two traditional approaches that aim to provide generic unicast routing mechanisms in such networks. AODV implements a *reactive* routing mechanism: routes are generated only when information is to be forwarded, reducing control message traffic but regularly introducing delays when sending packets. OLSR is an example of a link-state *proactive* routing mechanism, in which every node periodically maintains network topology information and routes for the whole network. In our use case, the number of nodes in the network is limited. Overhead due to proactive route maintenance, therefore, is not prohibitive. Different to forwarding routing decisions, we employ an acknowledgment mechanism that optimizes for robustness and prioritization, but we require detailed network state information including link quality, for which we utilize OLSR's topology information base [27, 28, 98]. We do not, however, use OLSR for actual information routing.

To determine best forwarding paths, we use the expected transmission count (ETX) metric [31], as simple hop counts would neglect differences in link quality, which may be particularly different in factory settings. The core idea of ETX is to estimate the number of transmissions that are necessary in order to successfully transmit messages. Thereby, ETX strikes a balance between link reliability – which short

links provide at the cost of a high hop count – and low delay – which can be achieved using (geographically) long hops at the cost of lossy communication. Draves et al. [34] compare different routing metrics for static and mobile network scenarios and argue that ETX performs best in static scenarios, such as our industrial use case.

Biswas et al. [13] propose an information dissemination protocol that is optimized for mesh networks and uses opportunistic overhearing to quickly cover large distances by adding meta-data called forwarding lists to each packet. In contrast to their proposal, we use overhearing during acknowledgment dissemination only, but we adhere to next-hop forwarding decisions for transmitting sensor information to reduce overhead and protocol complexity. Thereby, our protocol strikes a balance between the advantages of opportunistic overhearing, which can reduce the number of forwarding hops, and ad hoc routing, which removes the need for forwarding lists without unnecessary flooding of information and fully utilizes link-layer retransmissions without requiring hardware support.

In order to eliminate the need for complex routing path decisions and metrics, some works (e. g., [20, 72]) propose to apply network coding [5, 58] as an alternative that opportunistically disseminates coded packets instead of determining paths before transmission. While such network coding protocols can eliminate path calculation complexity, they may introduce additional network overhead, which can be prohibitive if network capacity is limited. Also, decoding a coded information collection requires receiving a sufficient number of independent linear combinations, disallowing prioritized and partial decoding. Existing approaches that allow prioritized decoding, e. g., [95], add significant computational complexity, which makes them unsuitable for most existing mesh networks. We revisit network coding and address the aforementioned problems in Chapter 5.

Related to generic mesh networks, several routing algorithms have been proposed to disseminate information using wireless sensor networks [66]. Often, these approaches use routing metrics similar to those discussed above but combine them with different requirements. Sensor networks require support for a much larger number of nodes, more frequent topology changes, and are bound by severe limitations on nodes' memory, computational capacity, and permissible energy consumption [7]. In our industrial setting, the number of nodes is usually limited, power is readily available throughout a workshop, and more powerful hardware can be used, rendering the faster but more expensive ad hoc mechanisms preferable.

4.3 *The TANDEM Protocol*

The TANDEM protocol operates on production cycles as defined in Section 2.6. Here, we present an overview of the protocol's operation before we delve into its details in the following subsections.

After a production cycle is completed, each machine transfers the acquired sensor information $\boldsymbol{a}_{(i,j,k)}$ to its attached wireless source node

n_i. Here, information is pre-processed and inserted into the nodes' transmission queue.[1]

[1] Pre-processing based on the discrete cosine transform (DCT) was discussed in more detail in Section 3.3.

Both source nodes and forwarding nodes schedule their information transmissions using a sending queue with an associated prioritization function. In the context of a multi-hop network, this function operates on information from different production cycles, different sensors, and different machines in the network. Nodes determine the shortest path to reach the sink using link-state routing information that containes information about the links' quality.

To improve robustness, acknowledgments are not only sent by the sink, but by each forwarding node on the path to the sink. To reduce overhead, these acknowledgements are sent periodically, acknowledging whole cycles instead of individual coefficient packets. Acknowledgments also leverage the wireless medium's inherent broadcast nature by implementing an overhearing mechanism. Responsibility for packet delivery is repeatedly passed on to nodes that are closer to the sink until eventually, the sink receives and acknowledges each packet.

Distributed information prioritization

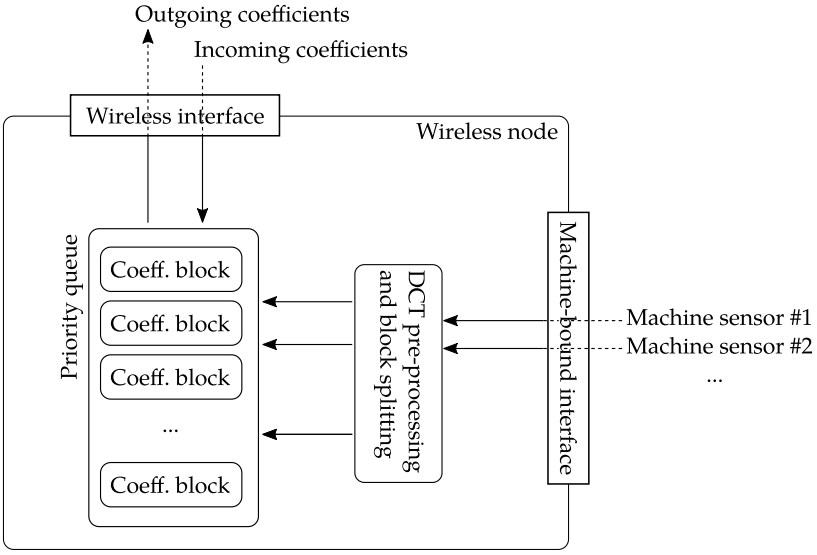


Figure 4.1: Input and output flows of a node's transmission queue.

Figure 4.1 schematically shows coefficient flows inside a single wireless node. Within a node, new process information enters the transmission queue in one of two ways. First, whenever the connected machine finishes a production cycle, this cycle's sensor information is pre-processed and inserted into the transmission queue, which is shown by the horizontal arrows. Moreover, in a multi-hop network, nodes may also act as forwarders to other nodes' information, which is the second option. Whenever a forwarding node receives packets with coefficients, these coefficients are inserted into the transmission queue as well.

Identical to the single-hop scenario described in Chapter 3, we sort the transmission queue by the coefficients' associated frequency, since

low-frequency components add more details to the preview of information at the sink. Due to the forwarding role of many nodes, the queue likely contains coefficient blocks from different machines in a multi-hop scenario. TANDEM nodes prioritize irrespective of the originating machine, that is, exclusively on coefficient frequency. Thereby, we ensure that all cycles can be approximated with low delay, and detailed representations are only transmitted when the current network capacity is sufficient.

Besides prioritizing information that provides an early approximation of sensor information, our transmission queue management also improves the fairness of network bandwidth allocation between machines. That is, it guarantees that important information is delivered independent of a node's distance to the sink – an important property that we discuss as part of our protocol analysis in Section 4.4.

Topology management and transmission

Whenever the sending queue is non-empty, the most highly prioritized coefficient block is forwarded towards the sink. To determine the next forwarding hop, each wireless node maintains the current network topology in form of a directed, weighted graph that represents link quality information. As we assume wireless nodes to remain at mostly static positions, we can obtain this topology information with tolerable communication overhead using existing link-state approaches.

The expected topology format is a graph that includes directed edge weights $w_i \in [0, 1]$ representing the expected packet delivery ratios. We obtain the ETX [31] for each directed edge – that is, the expected amount of total transmissions required to successfully transmit information along the edge – by taking the multiplicative inverse of the delivery ratio, $1/w_i$. Using the ETX-annotated graph as input, each wireless node then uses Dijkstra's weighted shortest path algorithm to determine the path towards the sink that has the lowest cumulative ETX. From the calculation results, the wireless node obtains its next hop towards the sink, which is determined by selecting the closest neighbor node on the calculated shortest path. In addition, the wireless node stores its own ETX distance to the sink:

$$D_{n_i} = \sum_i \frac{1}{w_{p_i}}, \tag{4.1}$$

where p_i are the edges along the node's shortest path towards the sink. Figure 4.2 shows n_2 as an example sending node, which obtains the marked shortest path towards the sink, a distance $D_{n_2} \approx 3.41$, and n_4 as its next hop. Note that all shortest path calculations are only used locally to determine the next hop; subsequent nodes perform the same calculations to determine their next hop, and so forth. No routing information is transmitted along with the coefficient blocks, because topology information is more recent in a node's proximity and wireless nodes can accommodate changing interference in the environment during transmissions along a path.

Once the wireless node has determined its next hop, it serializes the

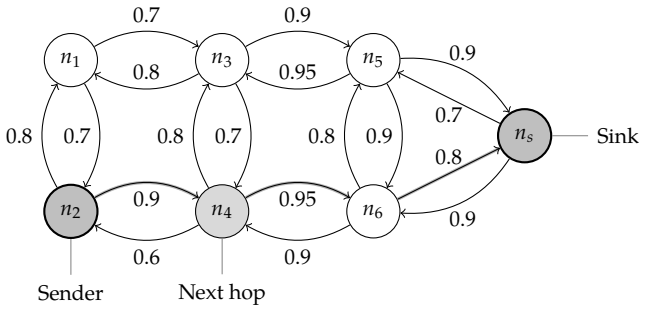


Figure 4.2: Topology management and next-hop calculation. © 2017 IEEE

next coefficient block b_i from its sending queue to a binary representation and transmits the packet to its next hop. Block b_i is then temporarily removed from the sending queue until it is either acknowledged by a wireless node closer to the sink (in which case it is removed permanently) or until it is considered lost (in which case it is re-scheduled for transmission).

Figure 4.3 shows the state machine that is used for re-transmissions, which incorporates timeouts for lost acknowledgment packets. The states can be summarized as follows:

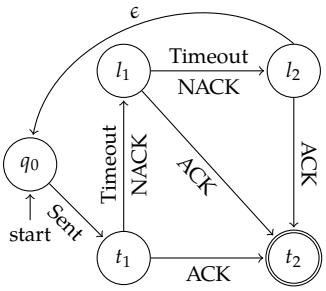


Figure 4.3: State machine used for re-transmissions. © 2017 IEEE

q_0 The coefficient block is in the transmission queue.

t_1 The block has been sent and is now in transit.

t_2 The block has been transmitted successfully.

l_1 The block is possibly/likely lost.

l_2 The block is considered lost and will be re-inserted into the transmission queue.

In the state machine, transitive state l_1 accounts for uncertainty if a given coefficient block was successfully transmitted or not: the transitive state l_1 can be entered even upon successful transmission if (a) the next-hop's acknowledgment message is encoded before the relevant coefficients were received or (b) if a timeout occurs after a positive acknowledgment has been lost. To avoid overhead, we thus only trigger re-transmissions from state l_2, which requires any two events of a timeout or a negative acknowledgment. Note that a single positive acknowledgment does not suffer from such uncertainty. All three states after initial transmission thus share an edge to t_2, which indicates successful transmission.

Different from end-to-end mechanisms, such as TCP acknowledgments, we implement reliability in a transitive manner. That is, packets that are being forwarded along a shortest-path segment only cause retransmissions on this segment and not, as it is the case for TCP, along the whole path.

Periodic broadcast acknowledgments

Each wireless node periodically transmits information until it receives an acknowledgment from another wireless node that is closer towards

the sink. We combine two optimizations to avoid unnecessary transmission overhead: first, each node acknowledges whole cycles at once and sends acknowledgements periodically. Second, nodes farther than one hop away may overhear acknowledgments to reduce re-transmissions.

An aggregated acknowledgment packet consists of the identifying machine-sensor-cycle tuple and a bit vector v. A bit 1 at position l in the bit vector indicates that coefficient block b_l of the cycle has been received. Conversely, $v_l = 0$ indicates that block b_l has not yet been received or was lost during transmission.

Acknowledgment periods are fixed time intervals. We say a machine-sensor-cycle is *fresh* if a node has received at least one packet with some of its coefficients during the current acknowledgment period. In every period, each node broadcasts an acknowledgment packet for every fresh cycle. The transmission using bit vectors saves bandwidth, because fewer, aggregated packets reduce the transmission overhead due to packet headers. Since acknowledgments are implemented as link-layer broadcasts, they can be overheard by all nodes within communication range, possibly skipping hops along the shortest path.

When nodes receive a broadcast acknowledgment, they determine its meaningfulness based on their own sending state machine, which is given in Figure 4.3, and their local topology knowledge: positive

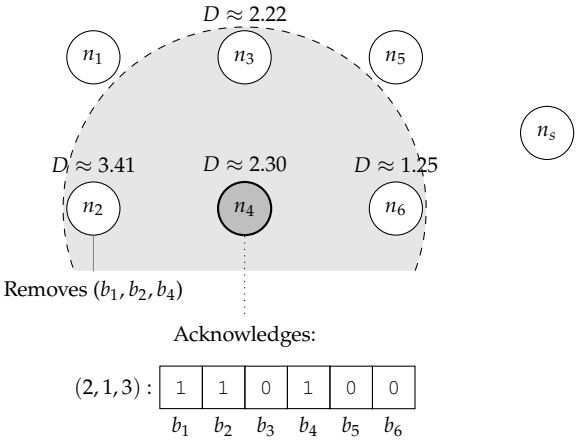


Figure 4.4: Opportunistic acknowledgments. © 2017 IEEE

acknowledgements are processed by all nodes farther away from the sink, irrespective of their next-hop relationships. That is, when a node receives a positive acknowledgement in form if a 1-bit, it compares its own cumulative ETX towards the sink to that of the acknowledging node. If its own distance is perceived as larger than the distance of the node that sent the acknowledgement by a margin δ, the respective coefficient block is considered to be successfully transmitted and is removed from the transmission queue.[2] In the example shown in Figure 4.4, assuming $\delta = 1/2$, n_2 removes b_1, b_2, and b_4 from its sending queue, because $D_{n_4} + \delta \approx 2.7 < D_{n_2} \approx 3.41$. The other receivers, n_3 and n_6, discard the acknowledgement, because $D_{n_4} > D_{n_3}$ and $D_{n_4} > D_{n_6}$, respectively.

In contrast to positive acknowledgments, negative acknowledgement bits only impact the sending state machine if the acknowledging

[2] The system parameter δ reduces the impact of temporarily conflicting topology information between nodes. Through experimentation, we found a margin of $\delta = 1/2$ (i. e., one half hop) sufficient to support reliability in all considered scenarios.

node had been selected as next hop for the particular block b_i. This restriction avoids superfluous re-transmissions, which otherwise occur whenever negative acknowledgements are overheard from nodes that are two or more hops downstream towards the sink and have not yet received the block in question. In Figure 4.4, n_3 and n_6 ignore all negative acknowledgements in the bit vector, because they never selected n_4 as next forwarding hop, whereas n_2 counts b_3, b_5, b_6 as negatively acknowledged.

4.4 Protocol Properties

This section discusses two protocol properties: reliability and fairness. With reliability we refer to guarantees that acknowledged information is delivered to the sink eventually. The fairness aspect is important to our use case as we want to make sure that all machines' information is prioritized equally, i. e., machines farther from sink are not at a disadvantage. Reliability is best evaluated analytically, since we are interested in worst case behavior; we assess fairness both analytically in this section and via network simulations in the next section.

Reliability

In Section 4.3, we argued that our hop-by-hop acknowledgment mechanism adds a high degree of reliability to the protocol, i. e., all confirmed information is delivered to the sink eventually for all practical purposes. As each node indefinitely re-transmits blocks to its respective next hop until an acknowledgment is received, with the final next-hop being the sink, the sink receives all information eventually when all nodes continuously execute TANDEM. In practice, however, unexpected node failure renders it impossible to guarantee reliability to the same degree as end-to-end protocols. That is, it is principally impossible to guarantee that all acknowledged information has been delivered with hop-by-hop protocol designs [70]. In protocols that acknowledge on a per-hop basis, already confirmed packets may be lost when (a) more messages are received than can fit in nodes' memory or (b) nodes exhibit failure.

Apart from insufficient memory and node failure, theoretical threats to reliability with hop-by-hop schemes include circular routing and, on a similar note, strongly diverging topology information with our opportunistic acknowledgment mechanism. Both threats can be mitigated by tracking the forwarding path for each packet and by fixing the next-hop for each production cycle. The mitigation impedes performance while adding little to reliability. If greater reliability is truly required for the particular use case, we thus suggest to consider redundant forwarding paths or complementary end-to-end reliability, since these techniques also mitigate failure of forwarding nodes.

In our system model, we assume that bottlenecks are temporary, which we argue avoids queue overflows with high likelihood. We model the duration of a node being a bottleneck, that is, a node receiving more information than it can send, with random variable $X\colon \mathbb{R} \to \mathbb{R}$. Since bottlenecks are temporary, the probability $\mathbb{P}(X > k)$ diminishes for a longer duration k and the limit $\lim_{k->\infty} \mathbb{P}(X > k)$ is 0; therefore,

$$\forall\, p \in (0,1]\, \exists k \in \mathbb{R}\colon \mathbb{P}(X > k) < p. \tag{4.2}$$

In other words, for any failure probability p, however small, we can find an upper bound k on the bottleneck duration. Even when assuming worst case behavior, in which case a bottleneck has a sending rate of zero, memory required to compensate for bottlenecks scales linearly

with the bottleneck's duration. Therefore, for each node, an upper bound $L \in \mathcal{O}(k)$ on buffer space requirements can be found with negligible error p. As a result, when network size, expected link rates, and each machine's sensor information rates are known, wireless nodes' storage size can be fitted large enough to guarantee sufficient buffering space for the use case,[3] which provides a high degree of reliabilty if nodes are functional at all times.

[3] A conservative upper bound on required memory can be found by muliplying all of these factors, which we exemplify below.

Further, we assume in our system model that node failure is temporary. While new information is routed around failed nodes via a different path, if available, some information may become unavailable upon node failure: when a node that temporarily stores acknowledged yet not forwarded blocks in its sending queue – and those blocks are not stored by any other node at the time – exhibits temporary failure, that node's blocks would be lost permanently if the information was only stored in volatile memory. In order to guarantee eventual information delivery, TANDEM nodes are therefore required to persistently store blocks in their sending queue and re-send them after recovering from temporary failure.[4] The delay of receiving the affected production cycles' process information will increase by the downtime of the node that exhibit failure, but information will be delivered eventually.

[4] Our protocol implementation also utilizes existing, volatile main memory without impeding reliability, which we describe in Section 4.5.

As an example, when nodes have 1 GB of persistent storage available for coefficient block storage, assuming a typical 500 Hz sample rate, 30 s cycle duration, four sensors per machine, and up to 15 machines in the workshop, TANDEM nodes can compensate bottlenecks and temporary node failures for at least two hours.

Fairness

The machines' sensor information is equally important regardless of whether a given machine is located next to the sink or at the other end of the production floor. Most end-to-end window-based protocols, such as TCP, however, result in fairness behavior that gives more distant nodes a smaller proportion of bottleneck link capacity [70]. What we strive for with TANDEM is to guarantee an identical share of bandwidth to all machines. In network literature, the term *max-min fairness* [11, pp. 524] has been used to to describe packet schedulers that only increase a packet flow's bandwidth when no packet flow with lower bandwidth can be increased instead. Hahne [51] showed that by employing a local round-robin scheduler at each forwarding node, the network eventually achieves global max-min fairness for every flow in the network given identical packet sizes.

The fairness analysis here is based on the classic approach inherited from wired networks, where fair bandwidth shares in ad-hoc networks are pre-assigned to a network graph [62]. Our system model differs, however, from such a wired scenario: (1) we have varying output rates depending on a node's current connection quality to its respective next hop, (2) we only allow for temporary bottlenecks, and (3), our input data rates are not constant, but cyclic. Despite these differences, we believe that the notion of max-min fairness is helpful to describe long-term behavior, where average output and data rates resemble constant rates, and we explicitly consider the impact of temporary bottlenecks.

Let $R(m_i)$ be the data rate from machine m_i's associated source node to sink n_s. Then, a feasible rate allocation vector $\boldsymbol{r} = \big(R(m_1), R(m_2), ...\big)$, i. e., a rate allocation vector for which no node exceeds its maximum sending rate, is defined as max-min fair if for each machine m_i, $R(m_i)$ cannot be increased while maintaining feasibility without decreasing $R(m_j)$ for some $j \neq i$, $R(m_j) < R(m_i)$.

Without any bottleneck nodes, i. e., no node's packet receive rate being greater than its maximum sending rate, TANDEM equals a

round-robin scheduler and therefore is max-min fair, as coefficient blocks with decreasing priorities arrive from each packet flow. Moreover, without bottlenecks, all rates in the rate allocation vector $\boldsymbol{r}$ must be equal.

Since our system model allows for temporary bottlenecks, next we consider the *aftermath* of such a congestion event, a non-equal rate allocation vector $\hat{\boldsymbol{r}}$, that is,

$$\exists\, i, j\colon R(m_i) < R(m_j). \tag{4.3}$$

We argue that after congestion events, our prioritization results in faster convergence to a max-min fair rate allocation than round-robin: forwarding nodes prioritize packets based on coefficient frequencies. We assume w. l. o. g. that forwarding node n_k lies on the shortest path of both flow $m_i \to n_s$ and flow $m_j \to n_s$, in other words, n_k schedules packets that pertain both to rate $R(m_i)$ and rate $R(m_j)$. After the congestion event, $R(m_i) < R(m_j)$ holds, therefore forwarder n_k will on average have more packets with low frequency components of m_i in its sending queue, which leads to a prioritized rate $R(m_i)$ for low-frequency components, accelerating convergence to max-min fair rates in the network. The same is less true for higher frequency components, since low-frequency components pertaining to $R(m_j)$ will be prioritized over high frequency components of $R(m_i)$, so restoration of max-min fair data rates takes proportionally longer for higher frequency coefficients.

4.5 Evaluation

To validate TANDEM's performance, we performed extensive discrete event network simulations. In addition and as a proof of concept, we implemented TANDEM on industrial grade wireless hardware and measured performance. Unless otherwise noted, data points in plots of this section show the arithmetic means and 95% confidence intervals. Error bars might not always be visible in the figures when the error is very small.

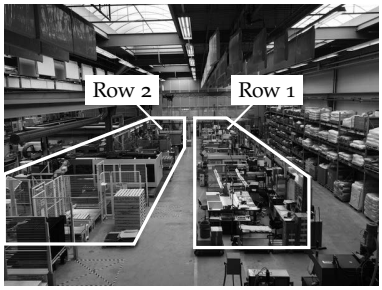
Methodology

Simulation set-up The simulation set-up closely follows Chapter 3: we base our simulations on the discrete event network simulator ns-3 [56] and model wireless transmission via YANS Wifi model [76] with IEEE 802.11g media access control (MAC) and 2.4 GHz physical layer. We consider obstructed line of sight and account for the effects of multipath propagation and large-scale path loss by employing Rayleigh and log-distance propagation loss models, one superimposed on another. To obtain a meaningful sample size, we repeat each simulation using 20 independent sub-streams of NS-3's MRG32k3a pseudo-random number generator (PRNG) [75]. In addition to the packet loss probability, we add randomness to simulations by varying machines' cycle start times and by adding a random delay before sending each packet.

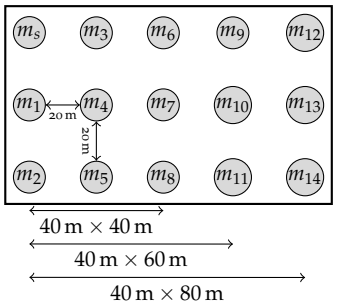
As time series data, we replay the pre-recorded sensor readings from real injection-molding machines and use packet sizes derived for a maximum packet size $P = 1024$ B. Identical to Chapter 3, nodes send with 1 Mbit/s rate and use a 200 ms acknowledgment interval unless stated otherwise. Each simulated machine has four sensors, two cavity temperature sensors and two pressure sensors. Each sensor records samples at 500 Hz rate over 25 s production cycle duration. After each production cycle, machines have a cool-down period twice as long as the cycle.

The simulated factory topology is based on typical industrial factory layouts. Figure 4.5a shows an example workshop that is approximately 50 m long and consists of two rows of injection-molding machines. To show scalability, we simulate a similar but larger three-row topology, as shown in Figure 4.5b and vary the factory length from 40 m to 80 m. Node distances are fixed to 20 m while network conditions are varied via the log-distance path loss exponent γ, using values sensible for factory environments [102]; for this topology configuration, Figure 4.6 maps γ-values to the packet delivery ratio of broadcast packets, i. e., without MAC-layer retransmissions.

We have described our evaluation data more closely in Section 3.6.



(a) A two-machine-row injection-molding factory.



(b) Simulation topology.

Figure 4.5: Simulation topology.
© 2017 IEEE

Topology information In Section 4.3, we described that each node requires up-to-date topology information, which is obtained using existing link-state approaches. To obtain up-to-date network topology information, each node thus runs an OLSR implementation. Figure 4.7 shows the interactions between TANDEM and OLSRd, the most common real-world OLSR implementation [98], in a single wireless node: OLSRd computes topology information by exchanging control traffic with other nodes. The TANDEM protocol obtains the ETX-annotated topology information from OLSR via a JSON-based querying application programming interface (API) provided by OLSRd. TANDEM regularly connects to its local OLSRd service via transmission control protocol (TCP) over a loopback interface. It then fetches and parses recent topology information and stores the ETX-annotated graph in memory.

The topology information from OLSRd is used in *both* the real-world implementation and our simulation. Ns-3 does provide a dedicated OLSR simulator model, but the model lacks important features compared to the real-world implementation. Most importantly, the simulator model does not provide the ETX information that TANDEM requires. The real-world OLSRd estimates ETX information via ''link-quality'' extensions, which also became part of RFC 7181 [27]. The ns-3 model, in contrast, only implements the original OLSR standard [28], which is limited to hop-count topology information. It is therefore necessary to load the real-world model to obtain accurate simulation results.[5]

To load the real-world protocol as part of the ns-3 simulation, we utilize a novel technique termed discRete event protocol emulatIon vesseL (gRaIL) [92]. Basically, gRaIL is a ''vessel'' – or wrapper – for binary protocol implementations. Combined with a specific protocol

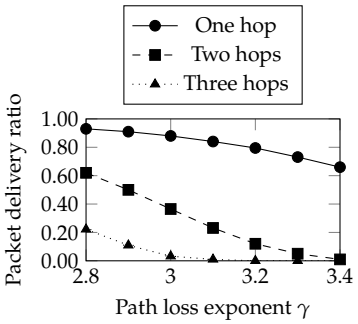


Figure 4.6: Expected delivery probabilities.

[5] Or to re-implement its extensions as part of the ns-3 model, which is a tedious and error prone task.

Figure 4.8: Prioritization effects on 40 m × 80 m factory floor ($\gamma = 3.4$).

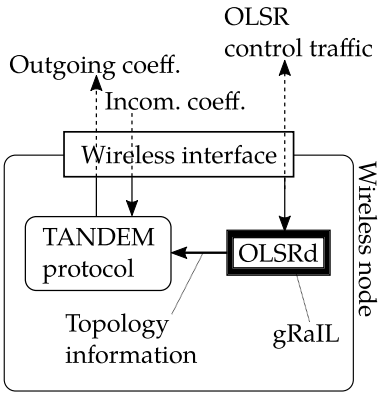


Figure 4.7: TANDEM and gRaIL in a single wireless node.

[6] We validate gRaIL in combination with OLSRd specifically in Section 6.7.

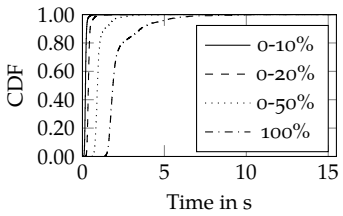


Figure 4.9: Prioritization effects on 40 m × 80 m factory floor ($\gamma = 2.8$).

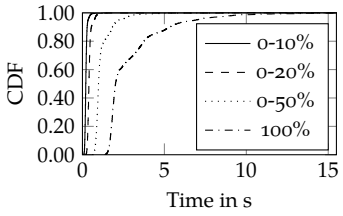


Figure 4.10: Prioritization effects on 40 m × 80 m factory floor ($\gamma = 3.1$).

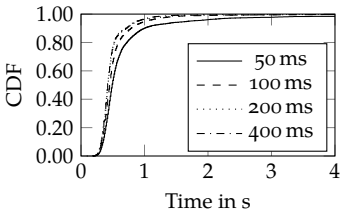
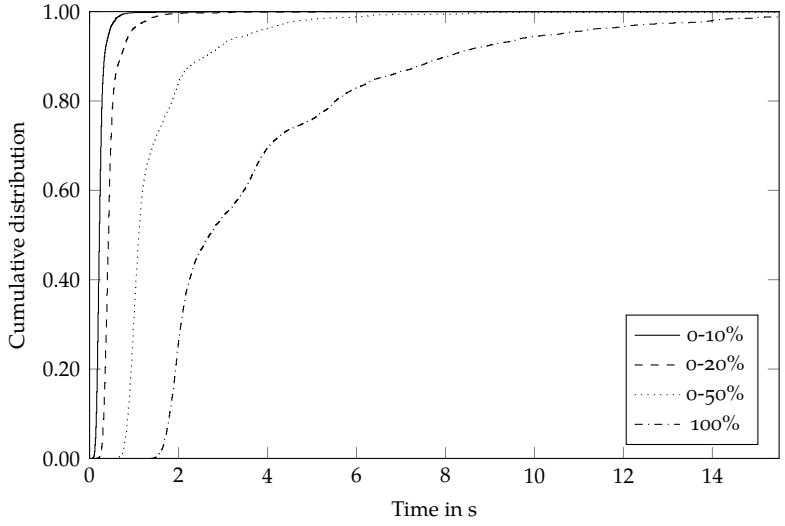


Figure 4.11: Effects of different acknowledgment intervals on 40 m × 80 m factory floor. ($\gamma = 3.4$)

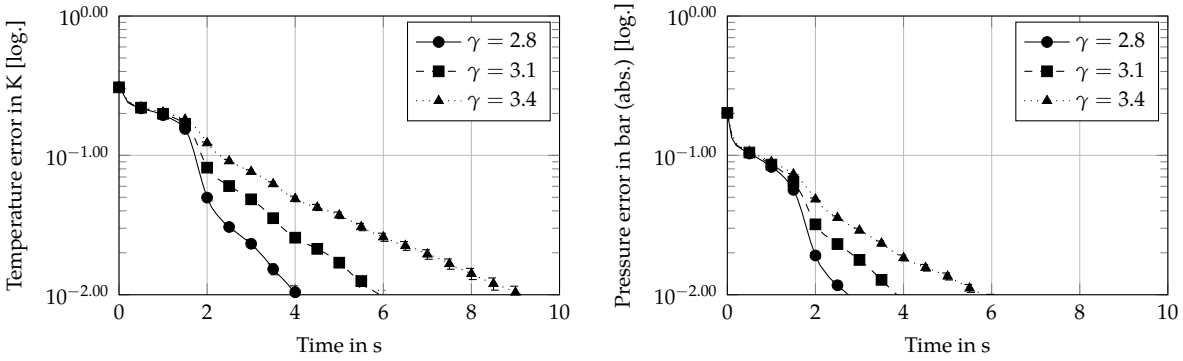


implementation, gRaIL provides an accurate discrete-event model of that protocol's logic and networking operations. The boundary between our normal ns-3 simulation and the emulated OLSR protocol, provided by gRaIL, is highlighted by the black box in Figure 4.7.

Since gRaIL is a complex software architecture and useful for a broad range of industrial protocol evaluation tasks, we describe it separately in Chapter 6. What is important here is that gRaIL in combination with OLSRd provides a *valid* simulation model of the real-world OLSRd protocol implementation.[6]

Prioritization effects

First, we evaluated how long it takes to receive a production cycle's prioritized low-frequency components. To this end, we simulated 15 machines in the 40 m × 80 m factory. Each machine transmits 8 production cycles over 600 s simulated time. Figure 4.8 shows the cumulative distribution function (CDF) of the relative time until the most prioritized 10%, 20%, 50%, and 100% of frequency coefficients were received. It can be seen that in-network prioritization has a drastic, non-linear effect: the most prioritized 10% and 20% of the lowest frequencies are available after a median time of only 220 ms and 420 ms, respectively, whereas median transmission duration of all the frequency coefficients is 2.7 s, which is one order of magnitude slower. The steepness of the low frequency components (10% and 20%) implies a high degree of fairness despite the very different distance betwen individual machines and the central server, which supports our fairness results in Section 4.4. A similar prioritization and fairness effect can be seen for lower path loss exponents (that is, better connectiviy) in Figures 4.9 and 4.10. We also examined the effects of different acknowledgment intervals on the prioritization performance, but overall found a low dependency of TANDEM on the acknowledgment interval. Figure 4.11 exemplary shows the relative delay to receive the most important 20 % of frequency components for acknowledgment intervals in 50 ms to



(a) Temperature error.

(b) Pressure error.

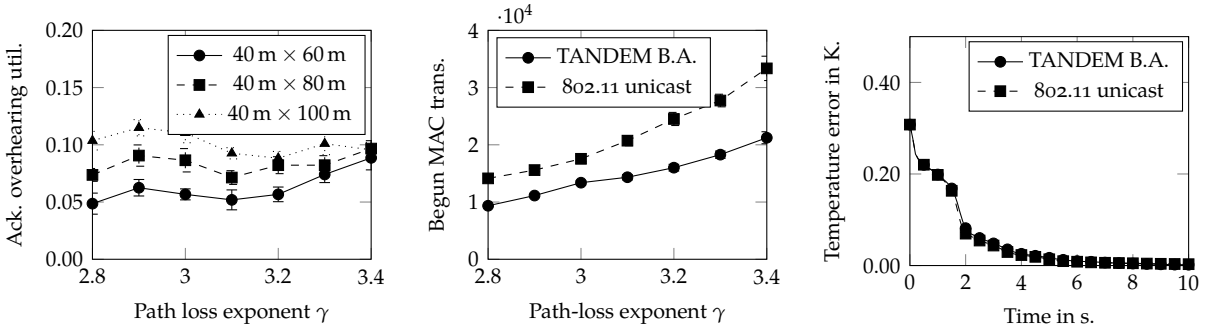
Figure 4.12: Average error over time for 40 m × 80 m workshop.

400 ms in the largest factory scenario with $\gamma = 3.4$. It can be seen that the receive times do not differ much, and are in fact worst for the 50 ms interval, which imposes the greatest probability to send non-innovative coefficients that were already successfully transmitted, as the timeout duration is derived from on the acknowledgment interval.

Prioritization and error

Next, we examined the effect of an incomplete set of coefficients on the sink's estimation quality. For this purpose, we use the pre-recorded sensor information from injection-molding machines that records both temperature and pressure within the mold. Pressure is given in bars (absolute) and temperature in degrees Kelvin. For each point in time we plot the average error based on the sink's information reconstruction. Here, we do not require that consecutive frequency coefficients were received, but instead also allow for gaps in the spectrum: analogous to missing high-frequency components, missing low-frequency coefficients are set to zero in order to reconstruct an approximation of the signal. Figures 4.12a and 4.12b show how the average error in the signal, that is, the average over all reconstructed samples' absolute error, decreases over time for temperature and pressure, respectively.

In our pre-recorded sensor data, pressure samples assume values between between 1 bar and 331 bar. For γ-values 2.8, 3.1, and 3.4, it took on average less than 100 ms until the error of the preview drops below 1 bar, which is already less than 0.5% of the value range. It took less than 0.6 s, 0.7 s, and 0.8 s, respectively, until the error of the preview drops below 100 mbar. Similarly, temperature values range from 306 K to 355 K and the mean error drops below 100 mK after 1.7 s, 1.8 s, and 2 s for γ of 2.8, 3.1, and 3.4, respectively. The average error drops slower for cavity temperature because samples are more distorted by noise, which results in less efficient DCT compression, necessitating more coefficients for precise decoding.



(a) Acknowledgment overhearing utilization. (b) MAC-layer transmissions of ACKs. (c) Precision comparison to unicast ($\gamma = 3.1$).

Figure 4.13: Broadcast acknowledgment mechanism.

Broadcast acknowledgment impact

Acknowledgment utilization: furthermore, we examined the extent to which TANDEM's broadcast acknowledgment algorithm helps in confirming packets based on overhearing. To this end, we let each node track whether blocks sent were confirmed via overhearing or by its designated next hop. Let h be the number of blocks that were confirmed by each node's next hop and o be the number of blocks that were confirmed by other nodes through overhearing, then we define the metric acknowledgment overhearing utilization as $o/(o+h)$. Figure 4.13a shows this average ratio for all nodes in the network as a function of path loss exponent γ. The acknowledgment overhearing utilization is shown for all three factory size configurations; it can be seen that the protocol benefits most from acknowledgment overhearing when packet loss rates are high. Since only paths that involve three or more nodes can add to overhearing utilization, larger topologies (40 m × 60 m and 40 m × 80 m) benefit more from the mechanism.[7]

[7] It can also be seen that the fixed distance between machines in the simulations makes some node distances more favorable for overhearing, explaining the somewhat wave-shaped function.

Acknowledgment savings: in addition, we evaluated how many transmissions were saved by the overhearing mechanism in comparison to standard 802.11 unicast, which involves MAC layer retransmissions. Figure 4.13b shows that TANDEM's overhearing mechanism consistently saves between 31.2 % and 57.1 % of MAC-layer transmissions for good and bad connectivity ($\gamma \in [2.8, 3.4]$). Figure 4.13c shows that using TANDEM's acknowledgment mechanism, the precision at the sink improves with similar speed as when using the unicast acknowledgment approach. For different γ-values, both acknowledgment approaches provide very similar performance. Thus, the saving in transmissions through TANDEM's acknowledgment mechanism, which means less energy used and less load on the wireless medium, usually comes without a cost in terms of preview performance.

Reliability: finally, we verify our analytic results on the reliability of our opportunistic hop-by-hop acknowledgment scheme. First, we note that for all simulation scenarios (all factory sizes and all γ in $\{2.8, 2.9, \ldots, 3.4\}$), we observed eventual transmission of all coeffi-

cients for 100% of all production cycles. In total, we observed more than 140 000 production cycles – or over 7 000 000 coefficient blocks – being successfully transmitted, confirming our analysis that the acknowledgment mechanism is highly reliable. On top of that, we tracked the size of the transmission queue, which we argued in Section 4.4 would be of limited size in scenarios without persistent bottlenecks. Figure 4.14 shows the queue size's cumulative distribution for the largest factory scenario and three different degrees of connection quality. We record the queue size in each node whenever it is about to transmit a coefficient block. The plot, thus, only accounts for times when a node is actively sending. Each production cycle produces sensor information from four sensors in parallel.

So even without any forwarding tasks, a queue size of more than 200 kB is reached regularly in each source node. In light of that fact, we consider the observed maximum queue size over all scenarios of about 1 MB low. If longer lack of connectivity is considered, i. e., longer-duration bottlenecks in Section 4.4, we expect the queue sizes to grow beyond these results, as described in the analytical results section. The figure also shows, as expected, that the queue size is somewhat smaller with better connectivity.

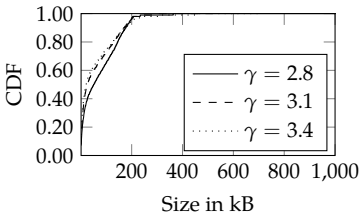


Figure 4.14: Cumulative distribution of nodes' transmission queue sizes at send events.

Real-world measurements

As a proof of concept and to validate that our simulation results correspond to real world scenarios, we implemented TANDEM on industrial grade IP68-certified hardware and evaluated the protocol in a four-node diamond shape topology with three source nodes and one sink node. Table 4.1 summarizes the main hardware and software components used.

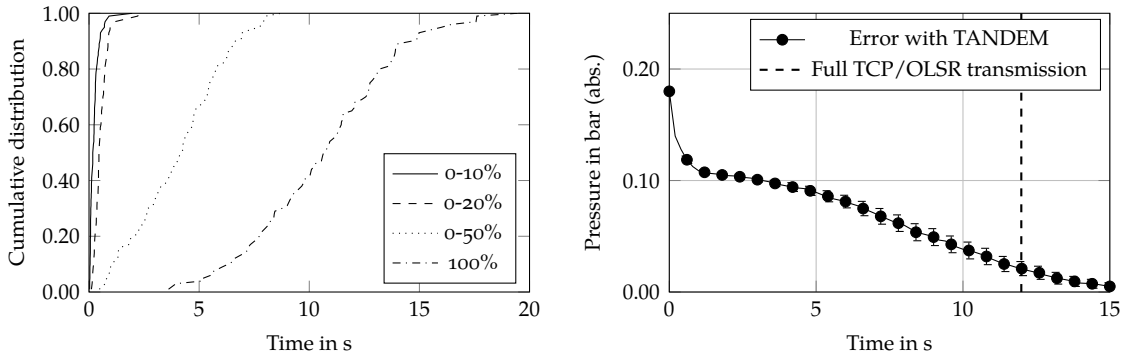
Table 4.1: Main system components. © 2017 IEEE

Component	Description
CPU	500 MHz AMD Geode LX800
RAM	256 MB DDR RAM
Storage	8 GB CompactFlash
Expansion	2 miniPCI slots
Ethernet	Via VT6105M
Antenna Ports	U.FL (Mini-SMT)
Wireless	Atheros XSPAN AR9220
Operating System	Debian 8
Impl. Language	Google Go

To support reliability, our TANDEM implementation employs a *persistent* database to store coefficient blocks that are recorded from a machine or received over the wireless link. Only an indexing key to the persistent database is written to the memory-backed prioritization queue that we described in Section 4.3. As a result, temporary node failure can be resolved on the next start up of the wireless node by iterating over all entries in the persistent storage and restoring the prioritization queue. The other two data structures, namely sender state and confirmation set, cannot be restored – which causes temporary redundant transmissions in case of node failures, but nonetheless ensures reliability.[8] Topology information is obtained via a loopback TCP link to OLSRd. The link is used to query OLSRd's JSON-based topology API, identically to the simulation scenario but without using gRaIL, as we are not running a simulation. Otherwise, the implementation follows Section 4.3, we well.

Prioritization results are shown in Figure 4.15a and confirm our simulation results: the prioritized information is transmitted quickly in all cases, whereas high frequency components take longer and are distributed less evenly. We further obtained a total of 20 comparison measurements using reliable TCP transmissions with equivalent

[8] Conceptually, the in-memory prioritization queue works like a cache. As soon as a coefficient block is received it can be forwarded from memory. Persistent storage becomes important when an acknowledgment is sent, as acknowledgments cause nodes to delete process information permanently.



(a) CDF of partial information reception.

(b) Mean error in pressure over time.

Figure 4.15: Real-world measurements.

data sizes, that is, including the relevant production cycle's meta data. Shortest paths were provided by OLSRd with link quality extensions enabled. TCP transmission took an average of 12.0 s, which is shown by the vertical line in Figure 4.15b. While TCP took less time for delivering all information, our in-network prioritization gives a close approximation much faster. As soon as a preview was available, the mean error we observed was lower than 180 mbar, which are less than 0.05% of the value range. Similarly, as soon as an approximate cavity temperature is available, its the mean error is less than 0.06% of the range.

In the real-world measurements, we observed an average acknowledgment overhearing utilization of 9.91% on the distant source node, which is the only node that can overhear acknowledgments due to the simple topology. Again, this result is in line with simulation results in Section 4.5.

4.6 Chapter Summary

To facilitate reliable wireless communication in larger-scale industrial environments, we described a wireless multi-hop transmission protocol, TANDEM, that is tailored towards the use case of delivering production processes' sensor information in a timely manner. Built upon the mechanisms in Chapter 3, TANDEM utilizes the DCT's high energy compaction to prioritize important information in the network and uses a novel hop-by-hop reliability mechanism that utilizes the wireless medium's inherent broadcast nature by overhearing acknowledgments.

Our simulative evaluation results, which are supported by a real-world implementation, show that our protocols' in-network prioritization ensures timely delivery of prioritized frequency components despite challenging network conditions. Using real sensor readings from sensorized injection-molding machines for evaluation, our evaluation shows that the error of an approximation based on our in-network prioritization is low and that this information is available much faster than delivery of all information would allow. We also observed that our acknowledgment approach saves transmissions by overhearing normally discarded, distant acknowledgment information.

5
Computing Within the Network

5.1 Overview

This chapter is based on previous work by the author [90, © 2016 IEEE], [94] and parts of a collaborative work [114, © 2018 IEEE]. In particular, the performance indicators described in Section 5.8 were derived jointly with co-author Marie Schaeffer and the iNsPECt extension of the network-coding implementation in Section 5.11 was developed by her. Preliminary investigations for Sections 5.8 to 5.11 of this chapter were conducted as part of a supervised study project and masters thesis [116, 117].

THE PREVIOUS CHAPTERS discussed mechanisms where the main computations were performed either at the source of information or – to some extent – at forwarding nodes. Limiting work performed by intermediate nodes can help to improve compatibility with existing networks, and, it bounds computational overhead at network nodes. After all, a node in a multi-hop network may forward much more information than it can produce in a given time frame.[1] The advent of more powerful industrial computers than what we assumed in Chapters 3 and 4, however, will enable more complex networking solutions.[2] Particularly, more powerful processing resources allow to perform computations on information that would otherwise only be forwarded.

An important approach to such computations is *network coding*, a widely studied and multifaceted approach to communication systems [42]. Network coding is, at its core, the act of combining multiple input packets to construct new output packets. Using network coding in our use case potentially improves the throughput, simplifies routing decisions, and adds robustness against packet loss – all using a single coding operation that is performed by all nodes in a distributed manner. In wireless multi-hop networks, this coding operation also implements opportunistic networking without requiring separate protocol mechanisms, rendering the protocols on top of network coding potentially less complex in design than traditional store-and-forward designs. In fact, challanges of wireless networks, such as unreliable links and packets' broadcast nature, which make traditional routing difficult, are characteristics for which network coding is a natural solution [32]. On top of that, network coding has been shown to be capacity achieving, that is, being able to achieve the maximum possible throughput in multi-hop networks, which is often greater than what traditional store-and-forward architectures can achieve [57]. Last, the approaches that we consider here naturally implement redundancy within the network, so that individual forwarding nodes can fail without impeding reliability.[3]

In this chapter, we explore to what extent the principles that we derived in Chapter 2 hold in the domain of network coding. Thereby, we bring about two important aspects: first, we shed light on a fundamentally different part of the protocol design space than what we explored so far; and second, we guide the transition of so-far-obtained results to future technology. To this end, we first give an overview of network coding and identify applicable coding schemes for our use case. Next, we identify two main challenges associated with such coding schemes, namely, efficient ''layer selection'' and efficient decoding capabilities. After reviewing related work, we propose two components that directly address these challenges: an optimized decoding technique and an overhead-minimizing layer selection mechanism. Last, we show how those (more fundamental) contributions can be fitted to the concrete injection-molding scenario from Chapters 2 to 4.

[1] Gupta et al. [50] showed that the ratio of ''own'' traffic versus forwarded traffic diminishes asymptotically for larger multi-hop networks, which is a fundamental result on ad-hoc networking scalability. The industrial wireless networks that we consider in this work, however, do not count the thousands of nodes that many internet of things applications demand. As a result, the asymptotic limitations are not fundamental to our use case, but demonstrate the potential pitfalls of applying expensive computations on forwarded contents.

[2] E. g., the industrial and embedded systems that we evaluated with in Section 4.5 were not capable of executing approaches that we consider here with sufficient performance.

[3] Source nodes must not fail permanently, being, at least temporarily, a singular location where production process information resides.

5.2 *Introduction to Network Coding*

This section introduces network coding and prioritized network coding. Here, we provide a high-level description of relevant operations, omitting some details; the following sections provide those details.

Details such as, e. g., how network coding ''messages'' are constructed and mapped to our use case or how decoding is implemented.

Traditional network coding

NC was introduced by Ahlswede et al. [5] to improve the throughput in communication networks. In their work, the network model is a directed graph with one node as the source and multiple nodes as receivers. Ahlswede et al. demonstrate that the optimal throughput, which is given by the "minimum cut" between the source node and any receiver in a network graph, can be achieved when the nodes send linear combinations of the original messages. Later, Ho et al. [57] showed that randomly chosen linear coefficients $c_1, c_2, \ldots, c_n$ over a finite field $\mathbb{F}_q$ are sufficient to achieve optimal flow rates in a wireless network setting; this approach is called random linear network coding (RLNC). With RLNC, linear combinations are built by multiplying the n original messages $\boldsymbol{m}^{(1)}, \boldsymbol{m}^{(2)}, \ldots, \boldsymbol{m}^{(n)}$ with the random coefficients, the j-th linear combination $\boldsymbol{x}^{(j)}$ being:

$$\boldsymbol{x}^{(j)} = \sum_{i=1}^{n} c_i^{(j)} \boldsymbol{m}^{(i)}. \tag{5.1}$$

In this chapter, we frequently employ double-index notation that is common in the network coding community. To help distinguish an index from exponentiation, we add parentheses to the upper index.

When multiple such combinations are received, they form a system of linear equations. The original messages can be retrieved by solving the system with, for example, Gaussian elimination, once a sufficient number of combinations were received.

Wireless ad-hoc networks benefit from applying RLNC, as it improves bandwidth use while reducing routing complexity [20]. As such, RLNC is an alternative to traditional routing-based approaches that require to calculate distance metrics between sources and sink and require to maintain routing tables [13].[4] Besides ad-hoc networks, the idea of network coding has been applied to peer-to-peer content distribution networks [3, 25, 134] and multimedia streaming [83].

[4] We have used such traditional routing information in Chapter 4 for forwarding decisions.

Prioritized network coding

A restriction of RLNC is that is not generally possible to decode a subset of messages with fewer than n linear combinations. However, all messages can be decoded immediately once enough linear combinations were received. This has been described as the "all-or-nothing property" [139] and is a fundamental restriction. In our scenario – where we aim to provide a detailed approximation of process information as early as possible – the all-or-nothing property is prohibitive as it nullifies any benefits from prioritization over the original messages. Our use case is not the only one in that regard: in peer-to-peer file sharing systems, and even more so in multimedia applications, the delay caused by waiting for enough linear combinations may be prohibitive for delivering sufficient service quality.

This limitation has been acknowledged early on in network coding research, e. g., Chou et al. [23] describe an information dissemination scheme for ad-hoc networks where more important information is coded with higher redundancy and can be decoded earlier by receiving nodes. Later, Nguyen et al. [95] propose prioritized network coding (PNC), which is based on RLNC and reduces per-packet delay. PNC introduces hierarchical layers $R_1, R_2, \ldots, R_{|\boldsymbol{R}|}$ of prioritized messages, where R_l ($1 \leq l \leq |\boldsymbol{R}|$) is a natural number that counts the number of messages contained in the priority layer l. Linear combinations of the l-th layer encode only messages from $m^{(1)}, m^{(2)}, \ldots, m^{(R_l)}$:

In literature, the encoding technique that we refer to as PNC here is also known as the expanding window random linear code [132].

Note that if just one priority layer is used, i. e., $\boldsymbol{R} = (n)$, Equation (5.2) equals Equation (5.1), and PNC equals RLNC

$$\boldsymbol{x}^{(j)} = \sum_{i=1}^{R_l} c_i^{(j)} \boldsymbol{m}^{(i)}, \quad \text{for a layer-}l\text{ combination.} \tag{5.2}$$

The main benefit of using prioritized network coding, i. e., applying Equation (5.2) over Equation (5.1), is that much fewer coded packets are required to decode all messages up to a particular priority layer.

Two important questions remain: first, how to determine which layer to choose for generating new linear combinations. That is, how to select l when generating a linear combination with Equation (5.2). The most basic approach, termed hierarchical network coding (HNC) [95], is to choose layers uniformly at random. HNC generally provides lower per-packet delay than RLNC, but increases the overhead due to non-informative linear combinations, i. e., linear combinations from layers that can already be decoded. Second, the question, how to decode such layered information can be implemented efficiently, has not seen much attention. If all layers are decoded using separate decoders, the amount of memory and the computational overhead required for decoding increases linearly in the number of layers, which limits the system's applicability. We address both of these questions with two separate contributions: an efficient, specialized decoder that has constant memory requirements and constant computational complexity with respect to the number of layers, and an integrated layer selection scheme that minimizes the overhead imposed by PNC.

5.3 *Related Work*

We have already introduced general network coding literature. In this section, we provide an overview of existing works that are closely related to our two main contributions.

Prioritized network coding

Esmaeilzadeh et al. [36] explicitly studied the layer selection problem both for systems without any knowledge about the receivers' decoder states and for systems with perfect knowledge about the decoder states. The proposed layer selection algorithm is based on an exhaustive search through packet erasures. In addition, finite-horizon Markov decision processes are proposed for the perfect-knowledge system model. The authors describe their perfect-knowledge model as idealistic, since perfect knowledge is usually unavailable. Additionally, both algorithms'

high computational complexity makes them unusable for practical applications with greater numbers of layers or sessions, but they may serve as a theoretical upper bound. We, different to [36], assume *limited* knowledge of the neighbors' decoder states and derive a simpler performance indicator that does not require exhaustive searching.

Shenglan Huang et al. [121] build upon HNC with uniform random layer selection to minimize the amount of redundant packets sent in a multi-sender use case. Their algorithm estimates, according to loss rates and information about links, the ideal number of linear combinations that each sender should produce to reduce linearly dependent combinations. The algorithm does not, however, provide layer selection capabilities different from HNC.

Chau et al. [21] also use the HNC coding technique but propose an additional coding scheme that combines messages from more than one HNC-coded generation. Thereby, the scheme provides additional redundancy, which protects against packet loss, and reduces the number of transmissions until all layers can be decoded. Our proposed layer selection technique could be used in conjunction with their HNC-based coding scheme, since it does not modify the coding format.

Approaches different from the layered PNC have been proposed to reduce per-packet delay in NC: Shrader et al. [122], for example, propose to employ a systematic coding approach. Namely, a subset of the network's nodes sends non-coded packets in some circumstances. Due to the selective choice of nodes, the level of error protection is not reduced, but the non-coded packets reduce per-packet delay as they do not require decoding. Yan et al. [139] instead trade correctness of decoded information for an increased chance of rank-deficient decoding by regarding the decoder matrix as a collection of underdetermined systems and implementing rank-deficient decoders. Finally, Claridge et al. [26] demonstrate that rank deficient decoding without chance of error is feasible when using a small enough finite field. Such a small field, however, also reduces the level of error protection and increases the chance of linear dependency. Different to PNC, all these approaches share that they do not guarantee decoding in order of priority, that is, it is possible that low priority information is decoded before high priority information. To highlight this difference, we use the term *per-layer delay* instead of per-packet delay, as the latter neglects messages' different priorities.

Decoding linear equation systems

Solving linear systems is necessary to retrieve original information in RLNC as well as PNC. Nodes transmit packets that are composed of an encoding vector, i. e., coefficients, and an information vector, i. e., coded information. Gaussian elimination [10] writes entries from the encoding and information vectors as rows into an extended coefficient matrix, the decoder matrix [43]. When the decoder matrix has full rank, the matrix is brought into upper triangle (i. e., row echelon) form and then solved using backward substitution. Gaussian elimination

requires $\mathcal{O}(n^3)$ finite field operations to bring a matrix into upper triangle form and $\mathcal{O}(n^2)$ operations for the backward substitution. Although RLNC reuses the coefficients for messages of b symbols each [47], both steps have to be performed b times, once for each symbol in a message. Rather than performing backward substitution once the matrix has full rank, it can be done incrementally when new linear combinations are received [25, 135]. Thereby, the total decoding delay is split over a longer time frame and less computation is necessary when the decoder reaches full rank.

LU decomposition is "a 'high-level' algebraic description of Gaussian elimination" [48, p. 111]. To solve the linear system $\boldsymbol{Ax} = \boldsymbol{z}$, LU decomposition works in two stages: first, the coefficient matrix $\boldsymbol{A}$ is factored into a lower and an upper triangular matrix $\boldsymbol{L}$ and $\boldsymbol{U}$, respectively, so that $\boldsymbol{A} = \boldsymbol{LU}$. Second, the utility system $\boldsymbol{Ly} = \boldsymbol{z}$ is solved by backward substitution and $\boldsymbol{Ux} = \boldsymbol{y}$ is solved by forward substitution. The first step takes $\mathcal{O}(n^3)$ finite field operations, the second and third step only require $\mathcal{O}(n^2)$ operations. Solving coefficients independent of information vectors is particularly beneficial for network coding, because encoding vectors are usually re-used for a number of symbols from the original information. In LU decomposition, the (computational complexity wise) more expensive first step has to be performed only *once* instead of b times; only the less expensive substitutions need to be performed b times. Therefore, we base our decoding algorithm on LU decomposition, which we extend to support decoding of prioritized layers in a joint decoding matrix.

5.4 System Overview

A network coding system consists of three main components: the encoder, the re-encoder, and the decoder [43]. We now explain how PNC can be applied to our industrial use-case, and we briefly cover all three components, reiterating necessary basics of network coding where necessary.

Information model

Mapping between production cycles and generations different than one-to-one mappings are possible, however, less beneficial: many production cycles in a single PNC generation increase delay before a preview is available. Conversely, distribution process information from one production cycle over multiple generations weakens error protection and diminishes the in-order decoding benefits of PNC.

In a network coding system, it is required that the information to be transmitted by a source node can be split into so called ''generations'' of equally sized messages $\boldsymbol{M} = (\boldsymbol{m}^{(1)}, \boldsymbol{m}^{(2)}, \ldots, \boldsymbol{m}^{(n)})$. In our use case, the concept of generations can conveniently be mapped to production cycles, so that messages within a generation are blocks of DCT-coefficients. Note that individual symbols in a messages are not necessarily equal to individual discrete cosine transform (DCT) coefficients. Each message $\boldsymbol{m}^{(i)}$ consists of b symbols $(m_1^{(i)}, m_2^{(i)}, \ldots, m_b^{(i)})$ over a finite field $\mathbb{F}_q$.

We are concerned with prioritized messages, i. e., some messages convey more information than others. In the following, and w. l. o. g., we assume $\boldsymbol{m}^{(i)}$ has higher priority than $\boldsymbol{m}^{(j)}$ for all $1 \leq i < j \leq n$. In case of DCT-coefficients, this criterion is trivially true if the messages

index reflects the frequency of the DCT-coefficients contained therein (as we assumed throughout Chapters 3 and 4).

PNC introduces hierarchical layers that we formalize as the vector $\boldsymbol{r} = (r_1, r_2, \ldots, r_{|\boldsymbol{r}|}) \in \mathbb{N}^{|\boldsymbol{r}|}$. Each entry r_l denotes the number of messages that the layer l adds, so it holds that $n = \sum_{l=1}^{|\boldsymbol{r}|} r_l$. Analogously, we define $\boldsymbol{R}$ as the cumulated layer vector with $R_i = \sum_{l=1}^{i} r_l$.

Encoding

To encode and transmit information, each source creates a sequence of network-coded packets; the j-th packet has the form $(\boldsymbol{c}^{(j)}, \boldsymbol{x}^{(j)})$. The components are:

1. an encoding vector $\boldsymbol{c}^{(j)} = (c_1^{(j)}, c_2^{(j)}, \ldots, c_n^{(j)})$, that is, a sequence of randomly chosen coefficients over $\mathbb{F}_q$, and
2. an information vector $\boldsymbol{x}^{(j)} = (x_1^{(j)}, x_2^{(j)}, \ldots, x_b^{(j)})$, which contains the actual linear combination.

The information vector of the l-th ($1 \leq l \leq |\boldsymbol{r}|$) layer and j-th coded packet is encoded as follows [43, 132]:

$$x_k^{(j)} = \sum_{i=1}^{R_l} c_i^{(j)} m_k^{(i)} \quad \forall 1 \leq k \leq b \tag{5.3}$$

Since finite field operations are generally performed over all symbols of a message or information vector, we usually omit the symbol index k, which reduces Equation (5.3) to Equation (5.2).

Multiple generations (i. e., production cycles) or multiple source nodes (i. e., machines) fit the above definitions by executing the encoding mechanism repeatedly in sequence or in parallel, respectively.

Re-encoding

PNC's re-encoding at intermediate nodes is identical to RLNC, but packets of different layers should not be mixed.[5] To create the q-th linear combination, a node re-encodes all applicable linear combinations of the l-th layer. We address these linear combinations by their indices, $o, o+1, \ldots, p$, so in total the l-th layer has $p - o + 1$ linear combinations. To build the linear combination, $p - o + 1$ random coefficients, named $\hat{\boldsymbol{c}}^{(q)}$, are generated. That is, one coefficient for each original linear combination. Next, each linear combination's encoding vector *and* information vector are multiplied by their respective, newly generated coefficients and cumulated. This process has been described as ''recursive,'' since it works analogously to Equation (5.2) [43]. The result, $(\boldsymbol{c}^{(q)}, \boldsymbol{x}^{(q)})$, is a valid coded packet of the l-th layer, as can be seen by the following transformation:

[5] Due to the hierarchical nature of layers, already-received linear combinations of higher-priority layers can be re-encoded as well. Encoding lower-priority linear combinations, however, can cause additional delays at the decoder side. Effectively, the newly generated linear combination has the same layer as the lowest-priority linear combination that it encodes.

$$x_k^{(q)} = \sum_{j=o}^{p} \hat{c}_j^{(q)} \cdot x_k^{(j)} = \sum_{j=o}^{p} \hat{c}_j^{(q)} \cdot \sum_{i=1}^{R_l} c_i^{(j)} m_k^{(i)}, \tag{5.4}$$

$$= \sum_{i=1}^{R_l} \underbrace{\sum_{j=o}^{p} \hat{c}_j^{(q)} c_i^{(j)}}_{=c_i^{(q)}} m_k^{(i)}. \tag{5.5}$$

Here, the leftmost expression shows the (''recursive'') construction rule for the information vector. The last expression demonstrates that this information vector indeed is a linear combination of the original messages generated using coefficients $\boldsymbol{c}^{(q)}$. Note that all original messages belong to the l-th layer.

The recursive generation of new linear combinations does not require decoding at intermediate nodes. In most protocols, however, intermediate nodes need to know (and report) whether received combinations were linearly dependent or not. Determining whether linear combinations are dependent works identically to decoding: the newly received linear combination is added to the decoder state, that is, the system of linear equations. If the rank of the decoder matrix increases after the application of elimination, the combination is independent. If the new linear combination's row in the decoder state was reduced to additive identities only, however, the row was linearly dependent.

The finite field $\mathbb{F}_{2^8}$ is a typical choice for network coding [43, 138] and is assumed in the following. With $\mathbb{F}_{2^8}$, linear dependencies between randomly generated combinations are unlikely [57], and the elements' binary representations occupy one byte each, which is advantageous in practical systems.

5.5 *Optimized Decoding Approach*

How early the receiver can decode the received information depends both on the time that the decoder requires for decoding and on the time until sufficient independent linear combinations could be obtained; the latter aspect depends on the strategy that each node uses to send coded packets of different layers. We now focus on efficient decoding aspects and afterwards discuss optimized layer selection strategies in Sections 5.8 to 5.11.

To accommodate for space efficient decoding of many hierarchical layers with low storage space overhead, we propose a new decoding algorithm, JOint laYer deCodEr (JOYCE), that is based on LU decomposition and a joint decoding matrix. Here, we provide an overview of our proposal's key algorithmic ideas. Section 5.6 will provide details of our implementation and a decoding example.

Assume the receiver collected g linear combinations: $(\boldsymbol{c}^{(1)}, \boldsymbol{x}^{(1)})$, $(\boldsymbol{c}^{(2)}, \boldsymbol{x}^{(2)}), \ldots, (\boldsymbol{c}^{(g)}, \boldsymbol{x}^{(g)})$. Each combination contributes a linear equation,

$$x_k^{(j)} = \sum_{i=1}^{n} c_i^{(j)} m_k^{(i)} \quad \forall 1 \leq j \leq g, 1 \leq k \leq b, \tag{5.6}$$

to b systems of linear equations, where b is the message size in symbols. Each of these systems has n unknowns, namely $m_k^{(1)}, \ldots, m_k^{(n)}$, and g knowns $x_k^{(1)}, \ldots, x_k^{(g)}$. A core requirement to decoding layered network coding is that the receiver must maintain its decoding state in such a way that, as soon as R_p linearly independent packets that encode the p-th layer have been received, the messages contained in that layer can be reconstructed.

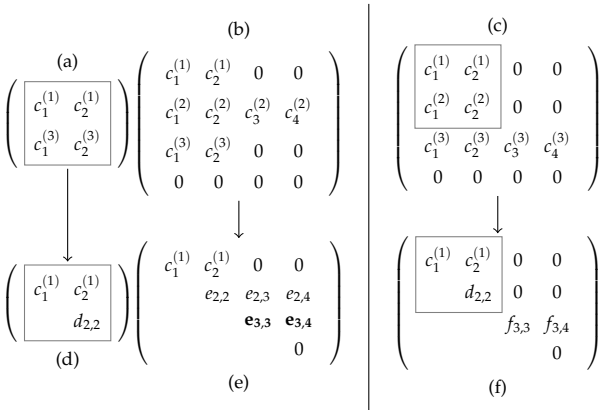


Figure 5.1: Decoder state: existing vs. proposed algorithm. © 2016 IEEE

To solve all k equation systems simultaneously, we use LU decomposition, as discussed in Section 5.3. As received packets may encode different layers of the original data, we need to modify the original LU decomposition algorithm to support joint decoding of different layers.

Figure 5.1 compares existing solutions and our algorithm in an abstract example that shows the upper triangular matrix of LU decomposition: the example considers two layers with two messages each $\boldsymbol{r} = (2, 2)$. The left-hand side shows the existing approach, with one matrix, (a) and (b), per layer. The right-hand side shows our approach, with only one matrix (c) for both layers. Arrows show how the matrices change when one iteration of Gaussian elimination is applied.

In the depicted example, the receiver has already received three linear combinations, two of which belong to the first layer and one belongs to second layer. The coefficients of these linear combinations' encoding vectors fill the first rows of matrices (a), (b), and (c). In the example's matrices, if the content of an entry is unknown, we introduce a new identifier named after its matrix, e. g., element $(3, 3)$ in matrix (f) becomes $f_{3,3}$. Whenever the contents of an entry is known, we use its more specific identifier, e. g., $c_2^{(1)}$ in all matrices.

The received information suffices to successfully decode the first layer, as two out of two linear combinations were received already. Without our approach, however, the receiver would need to maintain separate decoding matrices for each layer, in Figure 5.1 denoted by (a) and (b), which need to be solved in parallel, resulting in (d) and (e) after factorization was applied. Matrix (d) solves the first layer while (e) is the (so far incomplete) solution towards decoding the second layer.

Our contribution is to allow for early decoding of individual layers without the need for additional decoding matrices. Instead, we

regard the matrices for each individual layer as sub-matrices of a joint-decoding matrix (c). Without modification, applying LU decomposition to (b) destroys the different layers' encoding vectors' trailing zeroes. This effect is shown in (e). Intuitively, matrix (e) does not contain enough rows with additive identities in columns 3 and 4 to decode layer r_1, because trailing identities (entries $e_{3,3}$ and $e_{3,4}$) were overwritten during the elimination step. Therefore, the vector in row three of matrix (e) is no longer part of the linear subspace that corresponds to the first prioritization layer, and thus prohibits partial decoding. To enable joint decoding, we extend LU decomposition to maintain the *order of layers* during decoding, as shown in matrix (f), where entry $(2, 2)$ equals $d_{2,2}$ after factorization. Matrix (f) can be used to decode the first layer, while it is at the same time the currently optimal, incomplete solution towards decoding the second layer. The important lesson here is that joint decoding requires rows, i. e., linear combinations, that are *sorted by their respective priority layer*.

Note that the example in Figure 5.1 is a simplification since no decomposition has been applied to matrices (a), (b), and (c) previously. As a result, all the decoding work has to be performed at once. Our decoder, in contrast, works incrementally to evenly split the delay caused by decoding among the reception of individual linear combinations. Thus, it has to support partially decoded input matrices for the reordering step, which we describe in the next section.

5.6 Optimized Decoding Algorithms

Implementing the joint decoder matrix requires two main algorithms. First, we extend LU decomposition to allow for *incremental decoding* steps whenever new packets are received. Second, we implement a method we term *counter elimination*, which inverts individual steps of the LU elimination algorithm to allow for order preservation – and thereby early decoding – of individual layers. Third, we describe an optimization that introduces two utility structures ϕ and δ to refine the decoder's comparison function. This optimization, according to our analysis and experiments, has such a drastic impact on performance that we consider it an integral part of the decoding algorithm.

Initialization

A core aspect of LU decomposition is that the coefficient matrix is first solved independently before the solution is used to recover the original data packets. Our extension to that algorithm does not affect the remaining forward and backward substitution steps to recover the original data.

LU decomposition uses three matrices: a pivoting matrix $\boldsymbol{P}$, a lower triangular matrix $\boldsymbol{L}$, and an upper triangular matrix $\boldsymbol{U}$. Initially, $\boldsymbol{U}$ is the $n \times n$ matrix that consists of additive identities only, and $\boldsymbol{L}$ and $\boldsymbol{P}$ are $n \times n$ identity matrices.

When the first linear combination $(\boldsymbol{c}^{(1)}, \boldsymbol{x}^{(1)})$ is received, encoding

```
1: procedure UPDATE(c)
2:     u ← current matrix rank
3:     for all i ← 1, 2, ..., n do
4:         U_{u+1,i} ← c_i
5:     end for
6:     REORDERBYLAYER( )
7:     GAUSSIANELIMINATION( )
8: end procedure
```

Algorithm 1: Update decoder for new encoding vector $\boldsymbol{c}$. © 2016 IEEE

vector $\boldsymbol{c}^{(1)}$ is used to fill the first row in $\boldsymbol{U}$; $\boldsymbol{L}$ and $\boldsymbol{P}$ are not modified, and $\boldsymbol{x}^{(1)}$ is stored for later steps of the LU decomposition algorithm. No information can be decoded yet.[6]

[6] Except when $r_1 = 1$, in which case the received information trivially represents the first information block $\boldsymbol{m}^{(1)}$

Incremental decoding and reordering

Whenever an additional linear combination is received, we update the decoding matrices and incrementally decode the so-far-received information to ensure that information in prioritized layers can be decoded as early as possible.

The incremental decoding algorithm uses the decoding technique described by Fragouli et al. [43]. We extend the existing incremental approach by (a) modifying it for use with LU decomposition and (b) by maintaining a joint decoder matrix for prioritization layers $\boldsymbol{r} = (r_1, r_2, \ldots, r_{|\boldsymbol{r}|})$. To implement joint decoding, it is necessary that rows in $\boldsymbol{U}$, i. e., encoding vectors, are sorted by their associated prioritization layer. Otherwise, the trailing additive identities in their encoding vector would be overwritten during the elimination process, preventing the decoder from decoding prioritized data blocks early. For now, we focus on the implementation of incremental decoding using LU decomposition; we then discuss the details of the reordering process.

Algorithm 1 shows the high level decoder algorithm. The loop in lines 3 to 5 writes the new encoding vector in $\boldsymbol{U}$'s first free (i. e., ''all-zeroes'') row. Depending on the received packet's prioritization layer, $\boldsymbol{U}$'s rows may not be ordered by priority after the insertion. Therefore, we call the procedure REORDERBYLAYER (line 6) to reorder the rows according to their encoding vector's prioritization layer. Finally, Gaussian elimination (line 7) is used to eliminate elements in $\boldsymbol{U}$ and write the factors used to $\boldsymbol{L}$. As usual, elimination may perform row swaps as part of its pivoting, so the order of the rows may change again at this step.

Pivoting-row-swaps occur whenever a elimination step yields an additive identitiy on the main diagonal.

The procedure REORDERBYLAYER in Algorithm 2 implements a comparison-based in-place sorting algorithm. Essentially, the comparison operator $\blacktriangleleft$ is used to (re-)order $\boldsymbol{U}$ so that encoding vectors are ordered by the priority of their layers. When re-ordering is finished, the rows are ordered with the most important vectors at the top. Swaps perform the actual exchange of encoding vectors in $\boldsymbol{U}$ whenever out-of-order rows are found. Both comparisons and swaps need to take into account the particularities of LU decomposition to avoid side effects;

Algorithm 2: Adjacent-row-swap in-place comparison sort. © 2016 IEEE

```
1: procedure ReorderByLayer
2:     repeat
3:         s ← false
4:         for all i ← n − 1, n − 2, . . . , 1 do
5:             if ¬ (i ◀ i + 1) then
6:                 Swap(i, i + 1)
7:                 s ← true
8:             end if
9:         end for
10:    until ¬s
11: end procedure
```

Algorithm 3: Swap operation for rows i and $j = i + 1$. © 2016 IEEE

```
1: procedure Swap(i, j)
2:     u ← current matrix rank
3:     for all k ← j + 1, j + 2, . . . , u do
4:         CounterEliminate(k,i)
5:     end for
6:     SwapRows(P,i,j)
7:     SwapRows(U,i,j)
8:     for all k ← 1, 2, . . . , i do
9:         L_{i,k} ↔ L_{j,k}
10:    end for
11: end procedure
```

most importantly, they must update $\boldsymbol{L}$ and $\boldsymbol{P}$ to reflect changes in $\boldsymbol{U}$, and they must ensure that no additive identities are swapped to $\boldsymbol{U}$'s main diagonal.

To this end, we exploit that our decoder works incrementally; that is, whenever a packet is received, the matrix is in an almost-sorted state. When a new coding vector is added at the bottom of $\boldsymbol{U}$, it is likely out of order. Hence, we start sorting with the last non-zero row in $\boldsymbol{U}$. We use the encoding vectors' layers to decide whether two adjacent rows should be swapped. The layer information can be derived from an encoding vector by counting its number of trailing additive identities. The helper data structure γ_i maps row index i to its encoding vector's cumulative layer R_i for all $1 \leq i \leq |\boldsymbol{r}|$. Formally, two rows i and j where $i < j$ need to be swapped unless $\gamma_i \leq \gamma_j$:

$$i \blacktriangleleft j \Leftrightarrow \gamma_i \leq \gamma_j. \tag{5.7}$$

Note that Equation (5.7) implements the comparison function *without* the integral optimizations. The comparison function shown here will, without need, undo pivoting-induced row swaps at each step. We will update the comparison function to include the important optimizations in Equation (5.9).

Algorithm 3 implements the actual swapping of two adjacent matrix rows. The cell swaps in matrices $\boldsymbol{U}$ and $\boldsymbol{L}$ are performed in lines 6 to 10. But before doing so, the swap operation needs to ''undo'' certain elimination operations so that swapping rows i and j does not invalidate the decoder state. Most importantly, note that the row swap operation changes the entries in the matrix's main diagonal for rows i and j. Cells on the main diagonal may have been used for elimination, but $U(i,i)$'s new position may itself be subject to elimination after the swap. Therefore, lines 3 to 4 processes all cells that lie in column i and that lie

```
1: procedure CounterEliminate(i, j)
2:     for all l ← 1, 2, ..., n do
3:         U(i, l) ← U(i, l) + L(i, j) · U(j, l)
4:     end for
5:     L(i, j) ← 0
6: end procedure
```

Algorithm 4: Counter elimination. © 2016 IEEE

below the two swapped rows. For each of those cells, we apply what we term *counter elimination* to undo previous elimination operations and prepare the row swap.

Moving through $\boldsymbol{U}$ in decreasing row order (Algorithm 2 line 4), the comparison operation – and possibly the swap operation – is repeated with adjacent pairs of i, j where $j = i + 1$ until no out-of-order rows are found in the mutable part of $\boldsymbol{U}$.

Counter elimination

To implement counter elimination and ''undo'' certain elimination operations, we need to re-calculate previous values in $\boldsymbol{U}$ and alter $\boldsymbol{L}$ accordingly. Effectively, counter elimination is the partial reversal of a previous iteration of Gaussian elimination. The technique eliminates a factor in $\boldsymbol{L}$ and modifies $\boldsymbol{U}$ accordingly – whereas Gaussian elimination eliminates in $\boldsymbol{U}$ and modifies $\boldsymbol{L}$. Counter elimination, as shown in Algorithm 4, applies to a cell in $\boldsymbol{U}$ and $\boldsymbol{L}$ that is selected by row and column index i, j, respectively. In lines 2 to 4, the original values of $\boldsymbol{U}$ are restored using the information contained in $\boldsymbol{L}$. Afterwards, line 5 discards the corresponding elimination factor in $\boldsymbol{L}$.

Once counter elimination has been performed on all necessary cells, and all row swaps have been performed, the high-level algorithm continues to apply Gaussian elimination. Barring pivoting, $\boldsymbol{U}$ is still ordered by increasing layer priority afterwards and allows to decode information from all layers for which sufficient encoding vectors have been received. When receiving the next packet, the decoding algorithms are executed in the same way until all information layers have been recovered.

Optimized comparisons

In Equation (5.7), we defined the comparison function $\blacktriangleleft$ solely in terms of γ without delving into optimization details. Our experiments show that this definition, while functionally correct, results in subpar performance. Here, we explain necessary changes to the comparison condition to achieve high performance.

Gaussian elimination requires non-zero elements on $\boldsymbol{U}$'s main diagonal; when the elimination algorithm encounters a zero on the main diagonal, it performs row swaps as part of its pivoting. The simple swap condition $\gamma_i \leq \gamma_j$ reverts these pivoting swaps even if no rows that substantiate the need for pivoting changed. In other words, Algorithm 1 line 6 induces row swaps that are immediately undone in

line 7, rendering the swap operation useless in the first place.

Our optimized comparison avoids superfluous row swaps based on two utility structures ϕ and δ. The first utility structure, ϕ, is a marker that tracks which parts of $\boldsymbol{U}$ may be modified momentarily and which parts are immutable. Whenever a new encoding vector is received, ϕ is reset to point to the new vector's position in $\boldsymbol{U}$; all rows with a lower row index are considered immutable. At least one row in a swap operation must be mutable, otherwise the swap is skipped. Whenever two rows $i, i+1$ are swapped, we update $\phi \leftarrow \min(\phi, i)$. By adhering to these rules, rows are swapped only when the reordering is not necessarily undone by subsequent Gaussian elimination.

Skipping row swaps based on ϕ alone, however, introduces corner cases where our decoder cannot decode prioritized layers as early as possible. Specifically, pivoting may occur in the Gaussian elimination phase so that two encoding vectors i and j are not ordered by their respective prioritization layer. A newly received linear combination from layer l, with $R_l < \max(\gamma_i, \gamma_j)$, suffices (with high probability) to resolve such pivoting. Intuitively, the newly received linear combination must be swapped to a row position ''above'' the pivoting-induced out-of-order row. With ϕ-immutability in place, though, these out-of-order rows cannot be resolved (or: ''passed'') by new linear combinations that belong to a prioritization layer l with $R_l \geq \min(\gamma_i, \gamma_j)$.

We therefore define a second utility structure, δ, to ensure that pivoting can be resolved in all cases; δ maps a row index i to the maximum cumulative layer count found in all rows with an index smaller or equal to i:

$$\delta_i = \begin{cases} \gamma_i, & \text{if } i = 1 \\ \max(\gamma_i, \delta_{i-1}), & \text{otherwise.} \end{cases} \tag{5.8}$$

Based on ϕ and δ, we define the optimized comparison operation as follows:

$$i \blacktriangleleft j \Leftrightarrow j < \phi \vee \delta_i \leq \gamma_j. \tag{5.9}$$

Using δ instead of γ in the right inequality in Equation (5.9) enables swapping the new encoding vector above a point where pivoting occurred (i. e., with lower row index). Hence, an attempt is made to resolve pivoting during the next Gaussian elimination phase. This attempt is only made once, as the relevant vector is in the immutable region of the decoder matrix afterwards.

Decoding by example

Now that we have explained JOYCE's decoding details, we complement these algorithmic details with an example. Our example focuses on the interaction of the utility structures in particular: Figure 5.2 shows the main decoder matrix $\boldsymbol{U}$ in different decoding stages (a)–(g); $\boldsymbol{P}$ and $\boldsymbol{L}$ are not shown in the example. In addition to $\boldsymbol{U}$, we show the utility structures δ, γ, and ϕ left of each matrix. $\boldsymbol{U}$ is a 6×6 decoder matrix, but we only show the non-zero rows for conciseness. The dotted line

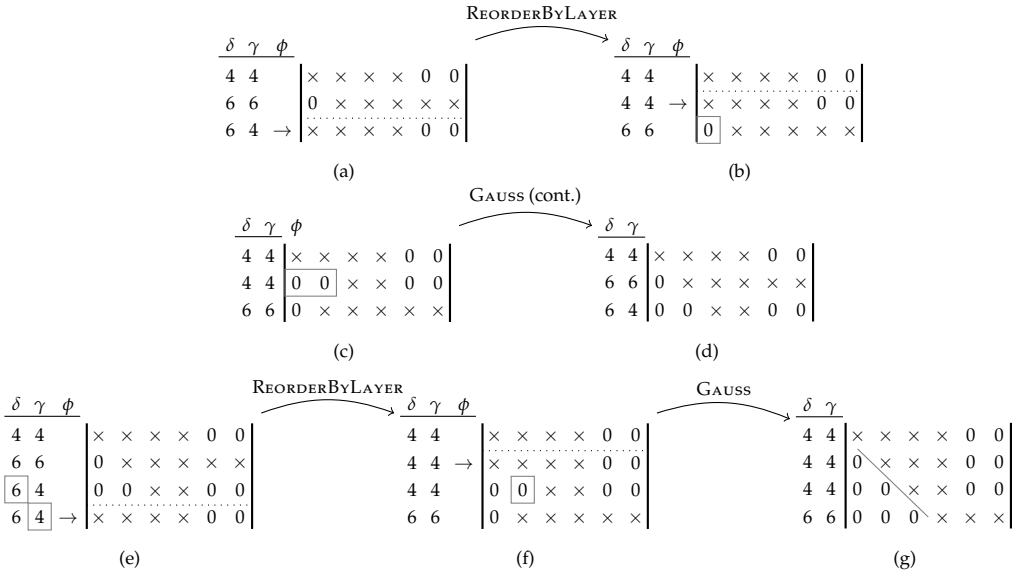


Figure 5.2: Decoding by example.

indicates that rows above this row index are set immutable by ϕ. We do not show the cells' exact content, but "0" for a finite field's neutral element or, otherwise, the placeholder symbol "×". The example coding system considers two layers, $r_1 = 4$ and $r_2 = 2$.

Initially, in state (a), the decoder previously received and processed two linear combinations, one from each layer. The third, newly received encoding vector is written to the third row and belongs to the first layer. Comparing δ_2 ($= \gamma_2$) to γ_3, the call to ReorderByLayer swaps rows 2 and 3. State (b) shows the result, in which order by prioritization layer is restored. In state (b), the already-eliminated cell ($U_{3,1}$) is still eliminated, i. e., the decoding work performed in previous increments was preserved by the swap. Next, Algorithm 1 line 7 invokes GaussianElimination. After eliminating $U_{2,1}$, $U_{2,2}$ also becomes zero in our particular example, as can be seen in state (c). Since $U_{2,2}$ lies on the main diagonal and must not be zero, Gaussian elimination invokes pivoting and reverses the previous row swap, which is shown in state (d), where the decoder matrix $\boldsymbol{U}$ is in upper triangular form.

In state (e), another linear combination of the first layer is received. Comparing δ_3 to γ_4 indicates that rows $(3, 4)$ need swapping in an attempt to resolve the pivoting. After swapping rows $(3, 4)$, the procedure ReorderByLayer swaps rows $(2, 3)$ and finally rows $(3, 4)$, resulting in decoder state (f). State (f) reveals that cell $U_{3,2}$, despite being subject to counter elimination, retains its zero value. After another application of GaussianElimination, matrix $\boldsymbol{U}$ in decoder state (g) is upper triangular. In addition, the matrix is ordered by prioritization layer. If another linear combination of the first layer is received, that layer can be decoded utilizing the first three rows' already-decoded state.

5.7 Evaluation: Optimized Decoding

To evaluate our algorithm, we assess it both analytically and measure performance of an example implementation. We compare decoding delay, computational complexity, and memory requirements of our algorithm compared to existing decoding without a joint decoding matrix. As discussed in Section 5.4, we assume a single sender and a single generation of information. Moreover, we allow for the sender to employ fine-grained layering (up to one layer per message); that is, $|\boldsymbol{r}| \in \Theta(n)$.

We will revisit layer selection strategies in Section 5.8. Here, we use a simple model that encompasses a range of reasonable strategies.

Finally, we assume that the sender uses a layer-selection strategy that selects layers for generating new linear combinations approximately in order of priority. Approximately, here, means that the sender may deviate from a perfect order by some offset, e. g., due to imperfect knowledge about the receiver's state. We model this assumption as a random deviation from the ideal order that is bounded by a constant C ($C \ll n$). Specifically, the sender transmits a number of linear combinations that each encode the currently highest-priority layer that helps to decode information, r_{ideal}, with a random offset uniformly distributed in the range $[-C, +C]$.

Analytical evaluation

We first discuss the *decoding delay.* Assume that the original information is divided into $|\boldsymbol{r}| \in \Theta(n)$ priority layers, and we are interested in the delay until the first layer's r_1 packets can be decoded. With RLNC, the decoder must receive at least n linearly independent combinations before it can decode the first layer. Using PNC, only r_1 linearly independent packets that encode the first layer are required. When linear combinations are sent in order of priority, PNC's decoding delay is in $\Omega(n/|\boldsymbol{r}|)$; if we assume fine-grained layering (i. e., $|\boldsymbol{r}| \in \Theta(n)$), partial decoding delay is in $\Omega(1)$. Using our optimized decoder does not change this property.

As described in Section 5.3, the dominant factor for *space complexity* during decoding RLNC is the $n \times n$ coefficient matrix; space complexity is $\mathcal{O}(n^2)$ [48, p. 116]. Decoding PNC layers separately requires $|\boldsymbol{r}|$ parallel decoders with a total space complexity in $\mathcal{O}(|\boldsymbol{r}| \cdot n^2)$. When using fine-grained layers, we have space complexity $\mathcal{O}(n \cdot n^2) = \mathcal{O}(n^3)$. Our approach uses a joint decoder matrix for all layers. The only additional data data structures are two mappings of size n from row index to prioritization layer, which are used for efficient reordering in the decoder matrix. These data structures are asymptotically insignificant compared to the space of the decoder matrix. Our decoding algorithm, therefore, has space complexity $\mathcal{O}(n^2)$, which is as good as RLNC and much better than PNC with existing decoding.

It is more difficult to find an upper bound on *computational complexity* than space complexity. LU decomposition requires $\mathcal{O}(n^3)$ finite field operations to factor a matrix of row vectors into $\boldsymbol{L}$ and $\boldsymbol{U}$. PNC performs n insertions of linear combinations into the matrices. Each

insertion, without our decoding technique, requires inserting into up to $|\boldsymbol{r}|$ decoding matrices, each taking $\mathcal{O}(n^2)$ finite field operations. PNC's computational complexity, therefore, is in $\mathcal{O}(|\boldsymbol{r}| \cdot n^3)$, which is $\mathcal{O}(n^4)$ for fine-grained layers.

To find an upper bound on computational complexity for JOYCE, we discuss the steps that are performed in addition to normal RLNC decoding. We take a top-down approach and start with Algorithm 1: JOYCE extends the incremental elimination process by a reordering step. This step takes place during an increment that is called $\mathcal{O}(n)$ times in total. Considering that we need $\mathcal{O}(n)$ increments, the total cost of JOYCE is $\mathcal{O}(n^3 + n \cdot C_R)$ where C_R is the cost associated with each call to ReorderByLayer.

ReorderByLayer is essentially a sorting algorithm that implements comparisons and swaps. In our implementation, comparisons are much cheaper than swaps. Comparisons require a constant number of operations, whereas row swaps consist of two for loops: one that performs the actual value swaps and one that performs counter eliminations. In the worst case, our reorder algorithm performs a quadratic number of swaps. However, we assume that linear combinations are received in order – respective to their layers' priorities – with a maximum deviation of C layers from the ideal layer. As sorting is only performed on the out-of-order part of the matrix $\boldsymbol{U}$, we have at most C^2 swaps, which is in $\mathcal{O}(1)$ with respect to n. The swap operation itself contains one loop that is iterated n times to perform the actual row swap. In addition, counter elimination is performed up to C times. Counter elimination itself iterates over all columns and therefore requires $\mathcal{O}(n)$ operations. This gives us $C_R \in \mathcal{O}(n)$ and a total computational complexity for JOYCE in $\mathcal{O}(n^3)$.

Algorithm	Delay	Space	Computations
RLNC	$\Omega(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$
PNC	$\Omega(1)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^4)$
JOYCE	$\Omega(1)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$

Table 5.1: Complexity overview.

Table 5.1 summarizes the results of the analytical evaluation in terms of delay, space complexity, and computational complexity. Using our decoder combines the best properties of RLNC and PNC: early decoding with low space complexity and low computational complexity. Of course, asymptotic complexity provides only a rough picture of the actual run time of algorithms, especially since sizes of n are often limited in practical network coding implementations. Hence, we complement our analytic evaluation with performance measurements.

Performance evaluation

We compare the decoding performance of JOYCE with existing, non-optimized decoding using separate decoding matrices per layer (hereafter referred to as ''PNC'' for consciseness, identical to the encoding process) and normal RLNC. Our implementations use the finite field $\mathbb{F}_{2^8}$ defined by the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$, as

suggested in [29]; hence, each input byte is uniquely identified by an element in the finite field. We implement multiplication as addition using a logarithm table and an ''anti-log'' table [29, p. 184]. All algorithms are implemented in C++ and compiled by Clang v. 3.7.1 using optimization level 3. We performed tests on a Linux system with kernel version 4.4.7, an Intel Core i7 processor, and 8 GB main memory.

To ensure that the process image and all libraries are in the file-system cache, we performed two preliminary runs before each measurement. The measurements themselves were repeated five times for each set of parameters. We refrain from implementing optimizations for the individual algorithms' intricacies to ensure comparability. All data points in plots show the arithmetic means and 95% confidence intervals (assuming normal distribution). Error bars might not always be visible in the figures when the error is very small. Test data and network coding coefficients were generated using C++'s `random_device` random number generator. Memory measurements show the *peak resident size*. Although we have implemented both stages to verify correctness, performance measurements only account for the time it takes to factor coefficient vectors into upper and lower triangular matrices. That is, only the asymptotically dominant factor, the first step of LU decomposition, is compared, since the second step is identical for all decoders and depends on the message size system parameter b.[7]

[7] Thus, when b is rather small, the decoding time shown is similar to the full decoding delay. E. g., in Chapters 3 and 4, we used $P = 1\,\text{kB}$, and $b \approx P$ would be a reasonable mapping here.

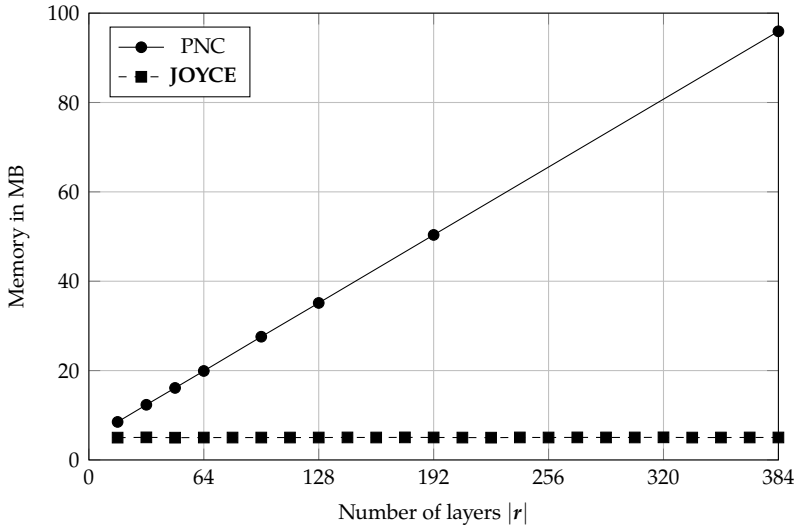


Figure 5.3: Decoding techniques' peak memory usage.

Memory usage: for memory usage, we fixed n and varied the number of prioritization layers, since those have a direct effect on the number of matrices for PNC. Memory usage for PNC and JOYCE is shown in Figure 5.3. The plot confirms Section 5.7's analytic results: PNC's memory usage grows linearly with the number of prioritization layers, whereas our approach's memory usage is independent of $|R|$.[8]

[8] PNC has fewer data points, as its implementation requires n to be a multiple of $|R|$.

Decoding time: as described in the analytical evaluation, the decoding time depends on the order in which the different prioritization layers'

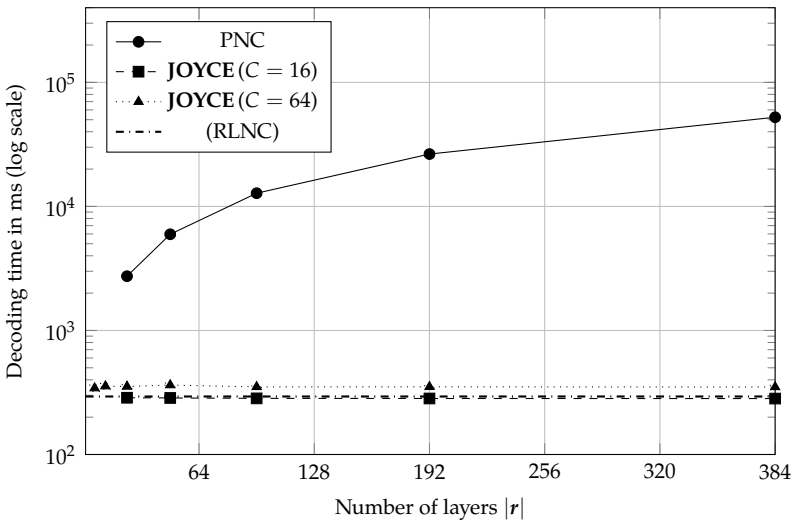


Figure 5.4: Decoder's cumulative, first stage decoding delay.

linear combinations are received by the decoder. In Figure 5.4, we use the same sender model as in the analytical evaluation and assume that the sender has only limited knowledge about the decoder's state, for example, due to packet loss. Since the layered prioritization approaches transmit more linear combinations than non-layered RLNC, we measure cumulative decoding time to keep the comparison fair. That is, we measure the time that *all* insertions take until full rank is achived.

Traditional PNC's decoding-performance is near-identical for different values for C; we exemplarily plot $C = 16$. The parameter $C = 16$ means that linear combinations encode a number of data blocks that differs from the ideal layer by up to 16 layers, which we consider sufficient for most applications. The parameter $C = 64$ yields an offset range of 128, which is already one third of n and represents extremely coarse information about the decoder state.

Our results show that, for both $C = 16$ and $C = 64$, our algorithm performs significantly better than PNC. Compared to RLNC, our approach runs 4% faster when $C = 16$ and 23% slower when $C = 64$. The first result is surprising, because RLNC does not use any prioritization layers. We would, therefore, expect RLNC to always perform better than JOYCE. We attribute our approach's performance gain to the simplified elimination process, as many of the cells in $\boldsymbol{U}$ are already additive identities when prioritization layers are used.

Finally, we evaluate performance in a worst-case scenario where the sender randomly chooses layers to use for creating linear combinations, i. e., follows the HNC layer selection strategy. That is, the sender's uncertainty is not bounded by C in this setting and, on top of that, we set $|r| = n$. We can see in Figure 5.5 that JOYCE's performance degrades significantly in comparison to Figure 5.4. However, even in this worst-case scenario, JOYCE performs consistently better than the traditional PNC decoder. We plot RLNC for comparison, even though no prioritization is supported.

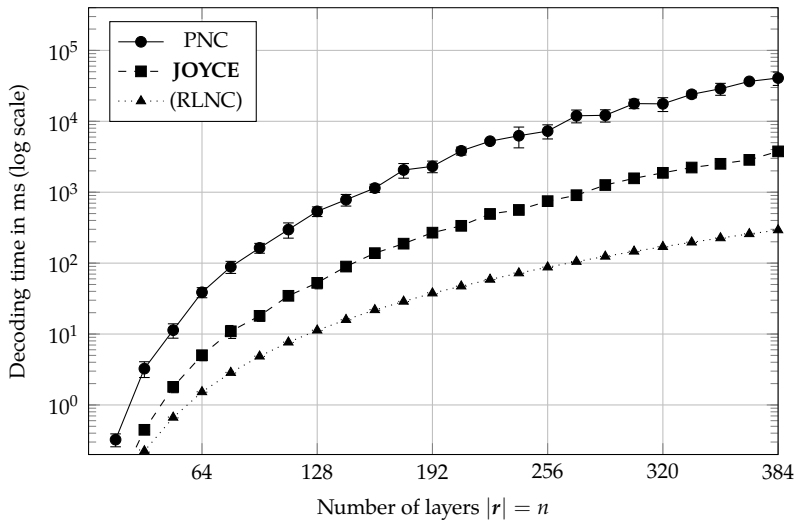


Figure 5.5: Worst case decoding delay.

Efficient decoding: evaluation summary

Summarizing, our measurements confirm the complexities derived in Section 5.7. Memory consumption of our approach is constant, whereas traditional PNC decoding has linearly increasing memory requirements with respect to the number of prioritization layers. The performance of our approach is also asymptotically better than PNC and similar to RLNC when we have limited knowledge about the sender state. This means that in most practical scenarios, our decoding algorithm can be used to add prioritization support to RLNC systems without incurring decoding cost, neither computationally nor memory wise. Even in the worst-case scenario, JOYCE performs significantly better than existing decoders, although in this case, prioritization comes at the cost of additional decoding complexity.

5.8 Layer Selection Problem

In the previous sections, we saw that the decoding performance of PNC largely depends on the layer selection strategy. During the performance evaluation, we already argued that it is a reasonable assumption to select layers for generating linear combinations approximately in order of priority. Here, we revisit the layer selection question and discuss principal properties of layer selection to determine how to efficiently select a layer for encoding. Besides impacting decoding performance, layer selection has a direct impact on (a) how long it takes until prioritized layers can be decoded due to sufficient decoder rank and (b) how costly prioritization is in terms of non-innovative linear combinations being sent.

According to our information model, data to be transmitted is partitioned into different priority layers. Given this partition, the challenge is to find an *efficient* transmission process implementing the prioritization scheme that is defined by the layers. Here, "efficient" comprises two aspects: low per-layer delay and low number of total transmissions.

The per-layer delay for prioritized layers describes the prioritization performance, whereas the total number of transmissions describes the overhead that is introduced. Ideally, both aspects are jointly optimized by *implementing effective prioritization without introducing overhead.*

It is impossible, however, to achieve low delay at the same time as low overhead in all situations, as these goals may conflict. Rather, we propose a strategy that jointly optimizes both aspects whenever possible and prioritizes low delay in conflict situations. To understand the connection between delay and total transmission number, consider an example topology with one source node S and three non-source nodes N_1, N_2, and N_3.[9] Assume a simple PNC system with a generation size $n = 4$ and two priority layers $\boldsymbol{r} = (2, 2)$. We will now discuss two scenarios: one where delay and overhead are in conflict and one where both can jointly be optimized.

[9] We employ a slightly different notation here to signify that network coding always considers *multiple* receivers of information. There is no single next hop, as in Chapter 4, but all nodes in the proximity receive and process linear combinations. It is therefore more helpful to describe examples for a single *sender* and multiple receivers than the other way around. Section 5.12 addresses the single-sink scenario more explicitly.

As the first scenario, assume nodes N_1, N_2, and N_3 have received $1, 1$, and 0 linearly independent combinations of the first layer, respectively, and only N_3 has received 1 linear combination of the second layer. Now, S sends two more linear combinations of the first layer. Then, N_1 and N_2 can decode the first layer after having received the first linear combination, and node N_3 can decode after having received both linear combinations. However, two more linear combinations of the second layer must be sent before all nodes can retrieve the second priority layer. If, instead, S immediately starts sending linear combinations of the second layer, all nodes must wait for one more transmission before they can decode the first layer. After only three transmissions in total (instead of four, as before), all nodes can decode layer one *and* two.

As the second scenario, consider nodes N_1, N_2, and N_3 have initially received $2, 1$, and 0 independent combinations of the first layer and $1, 2, 3$ independent linear combinations of the second layer, respectively. In this case, sending only one combination of the (lower priority) second layer from the beginning saves one transmission *and* achieves minimal per-layer delay.

The important observation here is that the goals of prioritization and not introducing extraneous transmissions *can* conflict, but this is *not always* the case. Our goal is to provide optimal prioritization first, but identify such occasions where we can save transmissions without introducing per-layer delay.

5.9 *Proposed Selection Algorithm*

In this section, we approach the problem statement with a simplified, theoretical model: nodes have knowledge of their neighbors' decoder matrix rank and each layer's linear subspace dimension. There are no packet losses or delays. We describe the proposed algorithm, eNhanced Prioritized nEtwork Coding (iNsPECt), from the perspective of an individual node. In Section 5.10, we incorporate packet losses, delays, and the need for feedback messages in the design of a network protocol based on this algorithm.

Our proposed algorithm combines two complementary strategies,

which we term "Ord" and "SL," that are used to decide which layer to select for transmission of the next linear combination. Both strategies (and our algorithm) make use of the fact that when one node's decoder matrix for a given layer has a higher rank than its neighbor's, sending a linear combination is with high probability innovative. "Ord," short for *in order,* is a greedy strategy that selects layers in strict order of prioritization; "SL" sends linear combinations of a *single layer* only. Our key idea is to use strategy Ord by default to minimize delay for prioritized layers. But we resort to sending a lower priority layer (with strategy SL) if it reduces the required total transmissions and does not negatively affect per-layer delay. Strategy SL takes a target layer i as a parameter; $\mathrm{SL}(i)$ sends only linear combinations of layer i until layer i – and thus all higher priority layers, as well – can be decoded. Obviously, $\mathrm{SL}(i)$ requires the minimum number of transmissions until layer i can be decoded; and strategy $\mathrm{SL}(|\boldsymbol{r}|)$ equals RLNC. Strategy Ord instead sends linear combinations of the highest priority non-decodable layer until each neighbor can decode that layer. It then continues with the next layer. Therefore, Ord ensures that high priority layers are always decoded before lower-priority layers.

To determine which strategy to use, our algorithm models the benefits of choosing one strategy over the other at any point in time with two performance indicators. Each indicator takes a parameter i and returns the benefits or drawbacks that result from choosing strategy $\mathrm{SL}(i)$ over Ord.

The first indicator, $Q_{\mathrm{rt}}(i)$, counts the *savings in total number of transmissions* until each neighbor of a node can decode layer i. Positive values indicate that using $\mathrm{SL}(i)$ is beneficial over choosing Ord. The second indicator, $Q_{\mathrm{dc}}(i)$, analogously counts the *additional per-layer delay* (in transmissions) until a node's neighbors can decode layers 1 to i, cumulated over the layers and all neighbors. Positive values indicate additional overhead introduced by choosing $\mathrm{SL}(i)$. In the following, we first derive the two indicators from our simplified system model and then define the selection algorithm.

Performance indicators

Reduction in transmissions: we define $\mathrm{RT}^{(*)}_{\mathrm{SL}}(i)$ as the number of required transmissions until all neighbors can decode layer i using the SL strategy. Analogously, $\mathrm{RT}^{(*)}_{\mathrm{Ord}}(i)$ denotes the number of required transmissions using the Ord strategy. Consequently, the savings in transmissions are:

$$Q_{\mathrm{rt}}(i) = \mathrm{RT}^{(*)}_{\mathrm{Ord}}(i) - \mathrm{RT}^{(*)}_{\mathrm{SL}}(i). \tag{5.10}$$

Next, we derive $\mathrm{RT}^{(*)}_{\mathrm{Ord}}(i)$ and $\mathrm{RT}^{(*)}_{\mathrm{SL}}(i)$. Let $\gamma_l^{(x)}$ be the number of independent linear combinations of layer l that neighbor x has received (and equivalently, the number of dimensions of the linear subspace that pertains to layer l), and let $\Gamma_l^{(x)}$ be the accumulated number of received combinations from layer 1 to l of neighbor x. Let $\mathrm{RT}^{(x)}_{\mathrm{SL}}(i)$ be the number of transmissions required for a single node x to decode

layer i. Layer i can be decoded once layer i's linear subspace has full rank, i. e., when $\Gamma_i^{(x)} = R_i$. Alternatively, layer i may be decoded when a lower priority subspace has full rank, i. e., $\Gamma_j^{(x)} = R_j$ for some $j > i$. Thus,

$$\begin{aligned} \mathrm{RT}_{\mathrm{SL}}^{(x)}(i) &= \min\left(R_i - \Gamma_i^{(x)}, \min_{j=i+1}^{|\boldsymbol{r}|}\left(R_j - \Gamma_j^{(x)}\right)\right) \\ \Leftrightarrow \mathrm{RT}_{\mathrm{SL}}^{(x)}(i) &= \min_{j=i}^{|\boldsymbol{r}|}\left(R_j - \Gamma_j^{(x)}\right). \end{aligned} \tag{5.11}$$

To generalize $\mathrm{RT}_{\mathrm{SL}}^{(x)}(i)$ to $\mathrm{RT}_{\mathrm{SL}}^{(*)}(i)$, we take the maximum over all neighbors:

$$\mathrm{RT}_{\mathrm{SL}}^{(*)}(i) = \max_{x \in \mathrm{Neigh.}} \mathrm{RT}_{\mathrm{SL}}^{(x)}(i). \tag{5.12}$$

We construct $\mathrm{RT}_{\mathrm{Ord}}^{(*)}(i)$ recursively. Since strategies Ord and SL are identical for $i = 1$, it holds that

$$\begin{aligned} &\mathrm{RT}_{\mathrm{Ord}}^{(x)}(1) = \mathrm{RT}_{\mathrm{SL}}^{(x)}(1), \text{ and} \\ &\mathrm{RT}_{\mathrm{Ord}}^{(*)}(1) = \max_{x \in \mathrm{Neigh.}} \mathrm{RT}_{\mathrm{SL}}^{(x)}(1) = \mathrm{RT}_{\mathrm{SL}}^{(*)}(1). \end{aligned} \tag{5.13}$$

Counting the required transmissions for the $(i+1)$-th layer, we first count the transmissions from the i-th layer and then add the remaining, maximum missing matrix rank over all neighbor nodes:

$$\begin{aligned} &\mathrm{RT}_{\mathrm{Ord}}^{(*)}(i+1) = \\ &\qquad \mathrm{RT}_{\mathrm{Ord}}^{(*)}(i) + \max_{x \in \mathrm{Neigh.}} \left(\mathrm{RT}_{\mathrm{SL}}^{(x)}(i+1) - \mathrm{RT}_{\mathrm{SL}}^{(x)}(i)\right), \end{aligned} \tag{5.14}$$

which, if we define $\mathrm{RT}_{\mathrm{SL}}^{(x)}(0) = 0$, reduces to

$$\mathrm{RT}_{\mathrm{Ord}}^{(*)}(i) = \sum_{j=0}^{i-1} \max_{x \in \mathrm{Neigh.}} \left(\mathrm{RT}_{\mathrm{SL}}^{(x)}(j+1) - \mathrm{RT}_{\mathrm{SL}}^{(x)}(j)\right). \tag{5.15}$$

Per-layer delay: analogously, we define $\mathrm{DC}_{\mathrm{SL}}^{(x)}(i)$ and $\mathrm{DC}_{\mathrm{Ord}}^{(x)}(i)$ as the cumulative per-layer delay (in transmissions) until node x can decode each layer up to i with the SL and Ord strategies, respectively. Similarly, $\mathrm{DC}_{\mathrm{SL}}^{(*)}(i)$ and $\mathrm{DC}_{\mathrm{Ord}}^{(*)}(i)$ define this delay cumulatively for all neighbor nodes. Our indicator,

$$Q_{\mathrm{dc}}(i) = \mathrm{DC}_{\mathrm{SL}}^{(*)}(i) - \mathrm{DC}_{\mathrm{Ord}}^{(*)}(i), \tag{5.16}$$

gives the additional per-layer delay that results from choosing strategy SL over Ord.

With the SL strategy, each node waits a timespan that is independent from the other nodes' decoder matrix states, as each node's rank of layer i increases independently until full rank is obtained. As the SL strategy only sends linear combinations of layer i, each non-decodable layer below i of neighbor x will become decodable after exactly $\mathrm{RT}_{\mathrm{SL}}^{(x)}(i)$ transmissions. Therefore, we count the number of non-decodable layers

(right-hand factor) multiplied by the number of transmissions required for decoding each layer (left-hand factor):

$$\mathrm{DC}_{\mathrm{SL}}^{(x)}(i) = \mathrm{RT}_{\mathrm{SL}}^{(x)}(i) \cdot \sum_{k=1}^{i} \min\left(\mathrm{RT}_{\mathrm{SL}}^{(x)}(k), 1\right), \tag{5.17}$$

$$\mathrm{DC}_{\mathrm{SL}}^{(*)}(i) = \sum_{x \in \text{Neigh.}} \mathrm{DC}_{\mathrm{SL}}^{(x)}(i). \tag{5.18}$$

Again, we derive the per-layer delay for the Ord strategy recursively and in two steps. First, we derive the non cumulated per-layer delay $\mathrm{nc}_{\mathrm{Ord}}^{(x)}(i)$ so that

$$\mathrm{DC}_{\mathrm{Ord}}^{(x)}(i) = \sum_{j=1}^{i} \mathrm{nc}_{\mathrm{Ord}}^{(x)}(j). \tag{5.19}$$

Again, for $i = 1$ both strategies behave identically, thus $\mathrm{DC}_{\mathrm{Ord}}^{(x)}(1) = \mathrm{nc}_{\mathrm{Ord}}^{(x)}(1) = \mathrm{DC}_{\mathrm{SL}}^{(x)}(1)$ and $\mathrm{DC}_{\mathrm{Ord}}^{(*)}(1) = \mathrm{DC}_{\mathrm{SL}}^{(*)}(1)$. For $i+1$, we distinguish between two cases based on whether the $(i+1)$-th layer can be decoded if the i-th layer can be decoded, which formally is the proposition:

$$\mathrm{RT}_{\mathrm{SL}}^{(x)}(i+1) = \mathrm{RT}_{\mathrm{SL}}^{(x)}(i). \tag{5.20}$$

If Equation (5.20) holds, the induction step is trivial, as no additional delay comes from layer $i+1$: $\mathrm{nc}_{\mathrm{Ord}}^{(x)}(i+1) = \mathrm{nc}_{\mathrm{Ord}}^{(x)}(i)$. If Equation (5.20) does not hold, node x requires exactly as many independent linear combinations as it is short of full rank to decode layer $i+1$, i. e., $\mathrm{RT}_{\mathrm{SL}}^{(x)}(i+1) - \mathrm{RT}_{\mathrm{SL}}^{(x)}(i)$. Since the Ord strategy will not start sending linear combinations of rank $i+1$ until *all* other neighbors can decode layer i, we also have to wait for $\mathrm{RT}_{\mathrm{Ord}}^{(*)}(i)$ transmissions before the $(i+1)$-th layer's rank increases:

$$\mathrm{nc}_{\mathrm{Ord}}^{(x)}(i+1) = \begin{cases} \mathrm{nc}_{\mathrm{Ord}}^{(x)}(i), & \text{if Equation (5.20) holds, else} \\ \mathrm{RT}_{\mathrm{Ord}}^{(*)}(i) + \mathrm{RT}_{\mathrm{SL}}^{(x)}(i+1) - \mathrm{RT}_{\mathrm{SL}}^{(x)}(i). \end{cases} \tag{5.21}$$

For all nodes, we obtain:

$$\mathrm{DC}_{\mathrm{Ord}}^{(*)}(i) = \sum_{x \in \text{Neigh.}} \mathrm{DC}_{\mathrm{Ord}}^{(x)}(i). \tag{5.22}$$

Algorithm

We have defined the two performance indicators $Q_{\mathrm{rt}}(i)$, which counts the required transmissions until layer i can be decoded by all neighbor nodes, and $Q_{\mathrm{dc}}(i)$, which counts the per-layer delay over all neighbors and layers up to i. We now use these indicators in an algorithm that takes a node's state as input and returns the layer choice as output. Whenever a node generates and transmits a linear combination, it executes the algorithm iNsPECt first, which is given in Algorithm 5.

To keep computational overhead low in practical systems, iNsPECt introduces a system parameter k_{ahead}, which bounds the number of

layers that a node may deviate from the Ord strategy. For large numbers of layers, bounding the deviation with k_{ahead} not only improves our algorithm's performance, but allows to use optimized decoding techniques [90].

Node state: $n, \boldsymbol{r}, \boldsymbol{R}$, and system parameter k_{ahead}
1: **procedure** LAYERCHOICE($\boldsymbol{\gamma}^{(x)}$, $\boldsymbol{\Gamma}^{(x)}$ $\forall x \in$ Neigh.)
2: choice $\leftarrow$ start $\leftarrow$ first non decodable layer of Neigh.
3: $\text{last}_{\text{rt}} \leftarrow Q_{\text{rt}}(start)$
4: **for** $i \leftarrow \text{start} + 1, \ldots, \min(\text{start} + k_{\text{ahead}}, |\boldsymbol{r}|)$ **do**
5: **if** $Q_{\text{rt}}(i) > \text{last}_{\text{rt}} \wedge Q_{\text{dc}}(i) \leq 0$ **then**
6: choice $\leftarrow i$
7: $\text{last}_{\text{rt}} \leftarrow Q_{\text{rt}}(i)$
8: **end if**
9: **end for**
10: **return** choice
11: **end procedure**

Algorithm 5: iNsPECt. © 2018 IEEE

In Algorithm 5 line 2, the Ord strategy is employed by default. That is, a node selects the layer with the highest priority that any of its neighbors cannot decode. To reduce the total number of transmissions, our algorithm uses the previously defined performance indicators in two steps:

(a) check if a lower priority layer can save transmissions by computing Q_{rt} and

(b) check if the lower priority negatively affects per-layer delay by computing Q_{dc}.

These steps are repeatedly performed in the conditional at line 5 for all k_{ahead} candidate layers in the loop at lines 4 to 9. Whenever a candidate layer in the loop further reduces transmissions compared to the last candidate *and* does not increase per-layer delay, it is selected as next candidate. Finally, the selected layer is returned in line 10.

Dependent on m, the number of neighbors from which feedback has recently been received, the algorithm's runtime is in $\mathcal{O}(m\, k_{\text{ahead}}\, |\boldsymbol{r}|)$. For three layers, a common choice in multimedia streaming [95], the runtime is linear in the number of neighbors.

5.10 Protocol Design

We now describe a simple yet effective network protocol that instantiates the layer selection algorithm for practical systems. We first describe the protocol for a single network coding generation and a single source, and then describe how the protocol supports multiple parallel generations (e. g., from multiple input sources) and subsequent generations (for continuous information delivery).

Message types

The protocol requires only two types of messages: data messages, which contain linear combinations, and feedback messages, which concisely encode a node's decoder state. We use UDP as the underlying transport protocol, because reliability is ensured by network coding's forward error correction properties.

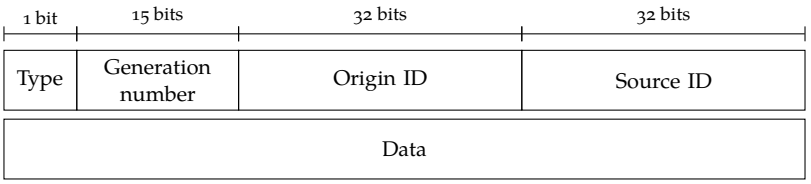


Figure 5.6: Message format: general header and message specific data part. © 2018 IEEE

The general message format is shown in Figure 5.6: a leading bit is used to denote message type, and the remaining 15 bits of the first two octets encode the generation number that the message belongs to. Origin node and source node, each encoded using 32 bits, are used to manage neighbor state and assign linear combinations to the correct decoding system. Each node is assigned a unique number in the system; alternatively, unique parts of the IP addresses can be used for local topologies.

Data messages contain linear combinations, which consist of an encoding vector and an information vector, as described in Section 5.4. If a linear combination from a layer with higher priority is sent, not all coefficients in the encoding vector are used; in that case, the remaining coefficients are set to the additive identity (i. e., "zeroes") of the finite field. Thus, encoding vectors contain n coefficients, whereas the information vector consists of b symbols. Each coefficient and symbol are elements of the finite field $\mathbb{F}_{2^8}$ and can be encoded as a single byte. Data messages, therefore, have a length of $b + n + 10$ bytes.

Feedback messages encode $\boldsymbol{\gamma}^{(x)}$; that is, they encode how many linearly independent combinations of each layer a node x has received. This is identical to the rank of each layer's decoder matrix or the dimension of each layer's linear subspace. A feedback message encodes each rank with two bytes; thus, the total feedback message length is $2 \cdot |\boldsymbol{r}| + 10$ bytes, where $|\boldsymbol{r}|$ is the total number of layers. Since the feedback messages' size does not depend on the generation size n nor on the message size b, feedback messages are usually much smaller than data messages.

Transmission mechanism

We employ a constant-rate approach to sending linear combinations. That is, if there is at least one generation which is not marked as fully transmitted every data message transmission interval λ_{data}, a node builds a linear combination according to Equation (5.3), encoded as a data message, and sends it. First, the iNsPECt algorithm is executed to determine the ideal layer i for building the next linear combination. If that layer-i's decoder matrix has insufficient rank to generate a linear

combination, i is incremented repeatedly until a linear combination can be built. If, initially, no feedback is available, we default to sending a linear combination of the highest priority layer.

Whenever a layer-i linear combination is sent, the feedback vector $\gamma^{(x)}$ for each neighbor x is incremented, presuming the linear combination's successful reception. In addition, a flag is set that indicates that the feedback vector is *assumed* instead of authoritative. If an assumed feedback vector indicates the generation is fully transmitted, the node continues to send linear combinations until an authoritative feedback message is received as confirmation. Thereby, nodes avoid delays at the end of each generation. By handling assumed and authoritative feedback in this way, we ensure that layers of lower-than-ideal priority may be selected, but never layers of higher priority. This bias may lead to increased per-layer delay for the current layer. It does not, however, cause additional overhead from linearly dependent combinations, nor does it affect the lower priority layers' decoding delay.

On reception of a linear combination, a node first counts the trailing number of additive identity elements in the encoding vector to determine the combination's layer. Next, the linear combination is inserted into each decoder matrix that pertains to a lower or equal priority than the linear combination's layer. Whether the newly received linear combination is innovative is determined using Gaussian elimination. If the rank of the matrix increases, the combination was innovative; otherwise, the new row is reduced to additive identity elements [43].

Feedback mechanism

Feedback is broadcast periodically and cumulatively for a generation's layers: once every feedback interval λ_{fb}, a feedback message is created for each incomplete generation. Usually, feedback messages are only sent when the generation is incomplete, i. e., the node's decoder matrix state does not have full rank for all layers. If, however, a linear combination for a complete generation is received, a feedback message that indicates successful reception of the whole generation (with $\Gamma_{|\boldsymbol{r}|} = R_{|\boldsymbol{r}|}$) is sent once.

The feedback's purpose is not only to inform other nodes about the current decoder state, but it also allows other nodes to learn about their neighborhood. When a node receives a feedback message from a node x, it includes x in its neighbor set (Neigh. in Algorithm 5) and updates $\gamma^{(x)}$ with the rank vector that is included in the feedback message. If a feedback message is received where the combination of origin identifier and generation number is unknown, that message is ignored, as the feedback contained is not helpful. Feedback vectors and neighborhood states expire after a timeout that should be chosen as a multiple of the feedback interval to avoid incomplete neighbor sets.

The proportion between the data interval λ_{data} and λ_{fb} is important for the performance of the proposed algorithm. The smaller the feedback interval, the better each node's stored feedback represents its neighbors' decoder state, since it is updated more often. On the other

hand, even tough feedback messages have a small size, more frequent feedback means more network capacity is used for traffic that does not directly contribute to the delivery of the sources' information.

Multiple generations

Often, information delivery is not a singular event but has to be performed on a regular basis. In this case, a new protocol state is created for each information generation. A fully-transmitted generation's state can be removed after a sufficiently long delay. The exact delay required depends on the use case's reliability requirements. At least, it should be larger than the feedback interval to account for badly connected nodes that might not yet have obtained full rank in their decoder state.

Parallel handling of generations is required whenever generations' transmissions overlaps. Overlap may be caused by challenging connectivity during the previous generation's transmission or by new generations that are spawned in parallel by one or multiple network nodes. When processing multiple generations, each generation's decoder state and feedback state are kept separately. Upon reception of a feedback message, the corresponding generation's state is identified by the unique tuple of generation number and origin ID. If no decoder state exists for that generation, an empty decoder state is created. The receiving node will now, as specified in before, send feedback messages for that generation as well to request linear combinations.

Before sending a linear combination, it is necessary to first select a generation from all currently active generations. Nodes select generations in a round robin fashion, prioritizing their own generations over those they forward for other nodes. The motivation behind this prioritized round-robin scheme is that a node's ''own'' generations have full rank by definition. Consequently, the chances of a node sending linearly dependent combinations that are derived from own generations is almost zero. But the chances are substantially higher for other nodes' generations. The earlier-described sending mechanism is then executed for the selected generation.

5.11 Evaluation: Layer-Selection

In this section, we briefly give an overview of the evaluation set-up and then provide detailed information on the simulations performed before we give evaluation results. Extended evaluation results that are more specific to the industrial use-case are given in the following section.

Methodology

Overview: we evaluate using the discrete event network simulator ns-3 (version 3.26) [56]. As in the previous chapters, wireless links between nodes are modeled via YANS Wifi model [76] with 802.11g MAC and 2.4 GHz PHY. We simulate the physical channel with the log-distance

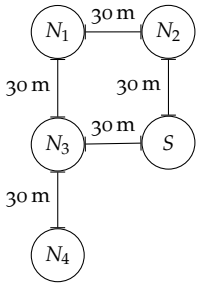
propagation loss model and the Rayleigh fast fading model, one superimposed on the other [54]. We choose the path-loss exponent $\gamma = 3.0$ and configured path loss at the reference distance 1 m according to Friis' model for 2.4 GHz, which is in line with a range of office and industrial environments [54, 102]. Nodes send actual linear combinations in the simulation, so there is a (small) chance for linear dependency even if a layer's decoder matrix does not have full rank. Each simulation is run for 600 s simulated time and nodes send five generations each. Simulations are executed 10 times; each repeated simulation uses a separate sub-stream of ns-3's MRG32k3a pseudo-random number generator to ensure uncorrelated results [75]. Pseudo-randomness is used in the simulation's wireless fading model, the ns-3 bit error model, which is affected by fading and path loss, generation of NC coefficients, and exponentially distributed variations in packet send times, which we use to avoid collisions and other synchronization effects. All figures in the evaluation sections show the sample mean and *95% confidence intervals* (assuming normal distribution). Error bars may not be visible when the confidence is very high.

Data input and output: simulated nodes model the arrival of information to be transmitted as a reoccurring event at a regular interval. Every i seconds, a new generation is created for each source node in the simulation. Each node creates five consecutive generations in each simulation run. We set $i = 5$ s for all small-generation ($n = 10$) simulations, $i = 25$ s for all larger-generation ($n = 50$) simulations. Linear combinations are sent by each node at a fixed output data-rate. The sending mechanism is implemented as a send event that reoccurs at a fixed output interval. That output interval is calculated by dividing the linear combinations' size by the output data rate. Single source, multi-source, and industrial simulations use 100 kbit/s, 200 kbit/s, and 1 Mbit/s output rate, respectively.

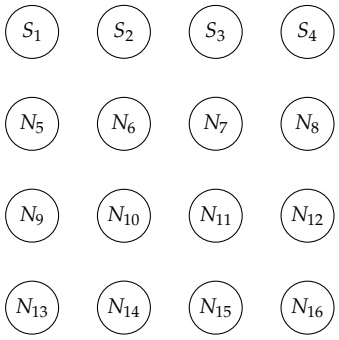
Timers in the simulation: to avoid synchronization artifacts due to deterministic timers, which may limit the results' validity, we implement all relevant timer intervals as (pseudo) random variables: the next send event for linear combinations follows an exponential distribution with expected value $1/\lambda$ being the output interval. For the feedback interval and the maintenance timer, we determine the duration until the next event by weighting only 50 % of the interval exponentially distributed and set the remaining 50 % fixed. This modification avoids bias from timers expiring too soon due to the exponential distribution's positive skew. We further avoid synchronization in data generation: each node initially delays data generation by a uniform random duration in the interval $[0, i]$.

Performance evaluation overview

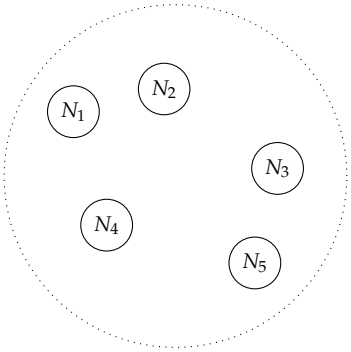
In this section, we focus on the layer choice's impact on decoding delay. To this end, we compare iNsPECt to PNC's HNC variant and



(a) Small regular topology.



(b) Larger regular topology.



(c) Wireless nodes in random disc topology. Here, only five nodes are shown, whereas the simulation uses 16 nodes.

Figure 5.7: Regular and randomized topologies used in simulations. © 2018 IEEE

RLNC, which we both described in Section 5.3. As a *lower bound* on decoding delay, we additionally show the decoding time for each layer, which results from sending only linear combinations of that particular layer. We note that this method to derive a lower bound uses smaller generation sizes, so the lower bound can be unattainable for full-sized generations.

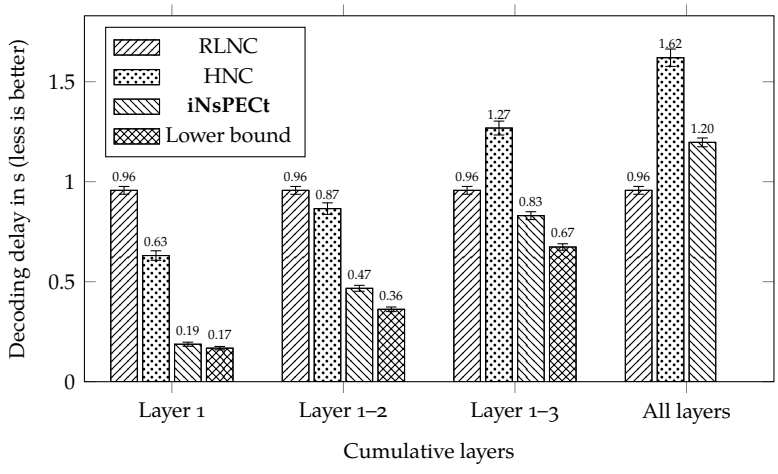
We highlight the impact on delay in three different simulation scenarios:

- a small-scale topology with a single source and small generations that comprise few source messages to evaluate the impact of varying feedback rates,
- a larger topology to show multi-hop capabilities and support for multiple sources, and finally
- a set of larger, randomized topologies to support the generality of our results.

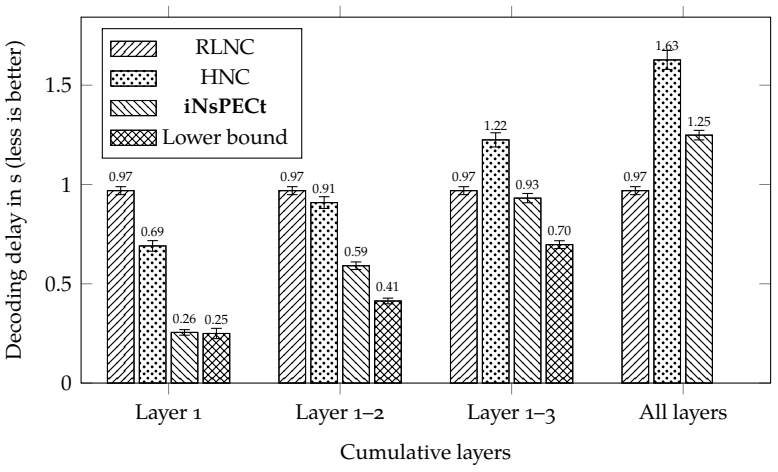
Small-scale topology

Figure 5.7a shows the small-scale topology: one source S and four nodes N_1 to N_4 are arranged in a partial grid with 30 m grid width. Due to the grid layout, nodes N_1 and N_2 have 30 m distance from the center-positioned source, whereas nodes N_2 and N_4 are 42 m away, which results in lower packet delivery ratio (PDR) for these nodes. We evaluate a small generation ($n = 10$) with small layers $\boldsymbol{r} = (2, 2, 3, 3)$ to highlight the effects of the layer selection and feedback rates.

The mean time until all sink nodes are able to decode layers 1 to 4, respectively, is given in Figure 5.8 for two different feedback rates. Results for frequent feedback are shown in Figure 5.8a: the y-axis shows the average time from beginning to transmit the current generation until a layer can be decoded by a node. The x-axis gives the individual layers, which are inherently cumulative due to the hierarchical nature of layers. It can be seen that using RLNC, the time until all layers can be decoded is identical for all layers. Since RLNC offers maximum protection against erasures and has the lowest chance to send linearly dependent combinations, it gives us the optimal decoding time for layer 4 (and thus all layers) in the last column. The HNC strategy, being fully randomized and independent of any feedback, provides faster recovery of the highest prioritized first layer, similar decoding time for the second layer, and worse delay than RLNC for the third and fourth layer. The proposed algorithm, iNsPECt, consistently outperforms HNC between 26.1 % and 70.3 %. Also, the delay is just 11.8 % higher than the *lower bound* for the most highly prioritized 1st layer (compared to 276.3 % for HNC). The layer 4 decoding delay of iNsPECt is only 25.1 % higher than the optimal RLNC delay. These 25.1 % analogously give the overhead imposed by the prioritization scheme, since the additional delay corresponds to the number of linearly dependent combinations that are due to prioritization. In comparison, this overhead is 69.3 % for HNC.



(a) *High* feedback rate ($\lambda_{\text{data}/\lambda_{\text{fb}}} = 1/1$)



(b) *Low* feedback rate ($\lambda_{\text{data}/\lambda_{\text{fb}}} = 1/3$)

Figure 5.8: Small topology results: per-layer delay for different feedback rates.

In a second step, we lowered the feedback rate to $1/3 \cdot \lambda_{\text{data}}$, which is quite low compared to the individual layer sizes: without losses, another layer has to be selected every two or three linear combinations or all subsequently sent combinations are linearly dependent. The results are shown in Figure 5.8b in a format identical to Figure 5.8a: RLNC and HNC, working without feedback, are unaffected by the change. iNsPECt is still faster than HNC for all layers, but due to the lack of recent feedback and the resulting uncertainty about the neighbors' decoder states, the algorithm resorts to sending layers of lower priority than necessary, which have a much lower chance of linear dependency. The downside of this approach can best be seen in the second and third layers, where decoding delay is still 35.0 % and 23.9 % better than HNC, but also increases by 26.5 % and 12.2 % compared to the better feedback rate scenario. A positive aspect of resorting to lower priority linear combinations is that linear dependency is less likely, which can be seen best at the fourth layer, where decoding time is not significantly different from the former scenario. This means that albeit iNsPECt's per-layer delay is not as low as before, it is still better than HNC *and* total prioritization overhead compared to RLNC does not increase at all.

Larger topology and random topology

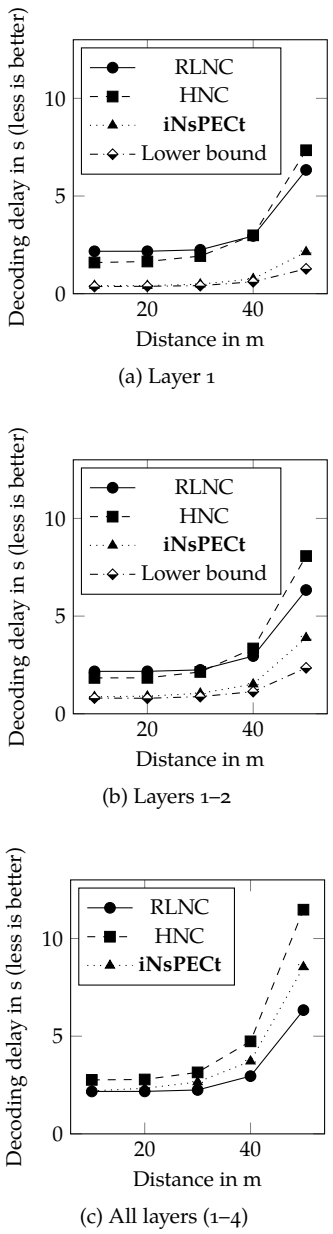


Figure 5.9: *Larger* topology results: per-layer delay for different (cumulative) layers and varying distances.

We now demonstrate scalability to larger ad-hoc networks with multi-hop requirements. The larger network topology is shown in Figure 5.7b and has 16 nodes in total: 4 source nodes and 12 non-source nodes. As before, nodes are arranged in a grid, but now we have four nodes in each row. We simulate a larger generation with $n = 50$ and 4 layers with layer sizes $\boldsymbol{r} = (10, 10, 15, 15)$. Figure 5.9 shows the simulation results for medium feedback rate ($\lambda_{\text{data}/\lambda_{\text{fb}}} = 1/2$) and varying grid distances x between neighbor nodes from 10 m to 50 m. Figures 5.9a to 5.9c give decoding delay for layer 1, layers 1–2, and layers 1–4, respectively.

iNsPECt allows nodes to retrieve the most highly prioritized 1st layer 73.9 %, 74.7 %, and 70.9 % faster than HNC for node distances of 10 m, 30 m, and 50 m, respectively. Also, the first and second layers' retrieval times with iNsPECt almost match their respective lower bounds for distances below 30 m.

Results look similar for the second layer in Figure 5.9b, where iNsPECt yields a 51.8 % to 52.5 % reduced per-layer delay over HNC. Interestingly, the benefit of HNC over RLNC for prioritized layers diminishes with greater distances between nodes (and thus lower PDR): at 40 m distance between nodes, HNC gives roughly the same per-layer delay as RLNC. We attribute the poor performance of HNC with low PDR in the large-scale scenario to inefficient multi-hop capabilities: when the source has few opportunities to successfully transmit linear combinations to the inner nodes in the network, low priority linear combinations are more useful, because they have a much higher chance of being innovative.

As expected, Figure 5.9c shows that the last layer – and thus all layers due to the hierarchical layer structure – is decoded the fastest with RLNC, which has the highest level of error protection against packet loss and the lowest chance of sending linear combinations of layers that already have full rank in neighbors. The delay given in Figure 5.9c is a direct indicator for the total number of transmissions required for sending one full generation. Therefore, it is also indicative for achievable throughput. For distances at or below 30 m, iNsPECt has at most 18.1 % overhead compared the optimum RLNC. This overhead increases to 25.8 % at 40 m distance between neighbors and 34.9 % at 50 m. HNC, in comparison, results in a message overhead of 81.1 % over RLNC.

Last, we verify the system's properties in a set of larger randomized topologies. Figure 5.10 shows mean per-layer delay for twenty different topologies where the nodes' locations are selected uniform at random in a disc that has the same overall area as the 30 m-distance grid topology. Again, we see a better performance of iNsPECt than HNC for all layers and a low total overhead of only 10.2 % compared to RLNC, which is the lowest overhead seen so far. Notably, iNsPECt enables decoding the most highly prioritized first layer 73.1 % faster than HNC.

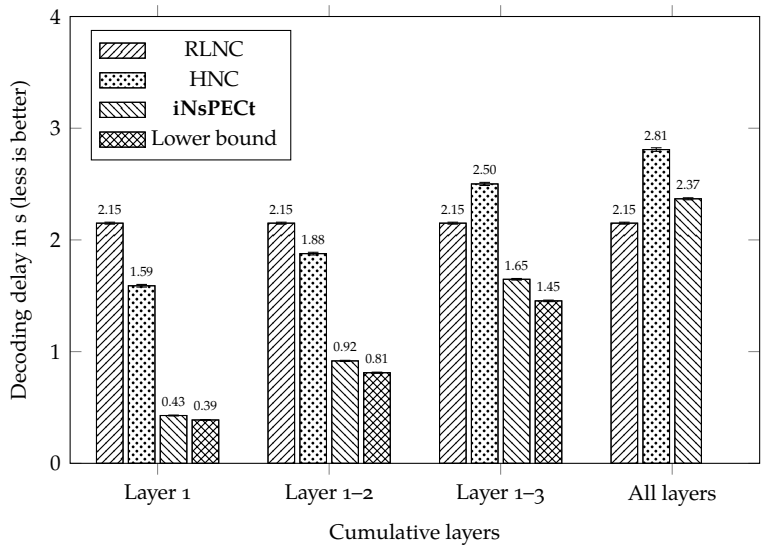


Figure 5.10: Random topology results: average per-layer delay.

Layer-selection: evaluation summary

We evaluated iNsPECt in both smaller and larger scenarios and compared it to HNC and RLNC for different feedback rates, distances, and topologies. iNsPECt significantly outperforms HNC in every scenario that we tested. Having only sporadic feedback and thus outdated decoder state information has no effect on the system's overhead in terms of linear dependency, but it increases decoding delay for some layers, albeit keeping delay significantly lower than HNC. Remarkably, in all scenarios, that is, for small and large distances, for high and low feedback rates, and for the small and larger-scale scenarios, the most highly prioritized layer's decoding delay was within 12 % of the optimum. Compared to RLNC, the overhead of iNsPECt in most regular grid scenarios was less than 25 %, which we consider a low cost for having prioritization. In non-regular, randomized topologies, iNsPECt has an even lower overhead that almost matches the lower bound on delay.

5.12 *Evaluation: Injection-Molding Use Case*

The last section considered layer selection benefits in terms of layer decoding delay for abstract networks. Here, we extend these results by considering the concrete, industrial example use case from Chapters 2 to 4 to demonstrate practical benefits of such a delay reduction. We also exemplify how network coding generation and protocol parameters can be derived from use case requirements so that network-coding-induced message overhead remains sustainable.

As we recall, in our example use case, plastic injection molding, machines inject molten plastic with high pressure and temperature into a form, termed the ''mold.'' In this section, we compare the impact of the two prioritized network coding techniques HNC and iNsPECt on delays in sensor data collection. As a third mechanism, regular RLNC [57] serves as a baseline for our comparison.

Building generations from cyclic sensor information

Using regular RLNC as an example, we explain how network coding in general can be applied to our sensor data collection use case. We then discuss how prioritized network coding mechanisms can be combined with the presented industry use case.

RLNC splits information into generations of data messages. In our case, a generation can be mapped to one *production cycle* worth of sensor information from a *single sensor,* i. e., the message set $\boldsymbol{M}$ for a cycle (i, j, k) holds $\boldsymbol{a}_{(i,j,k)}$. As before, each message consists of symbols over the finite field $\mathbb{F}_{2^8}$.

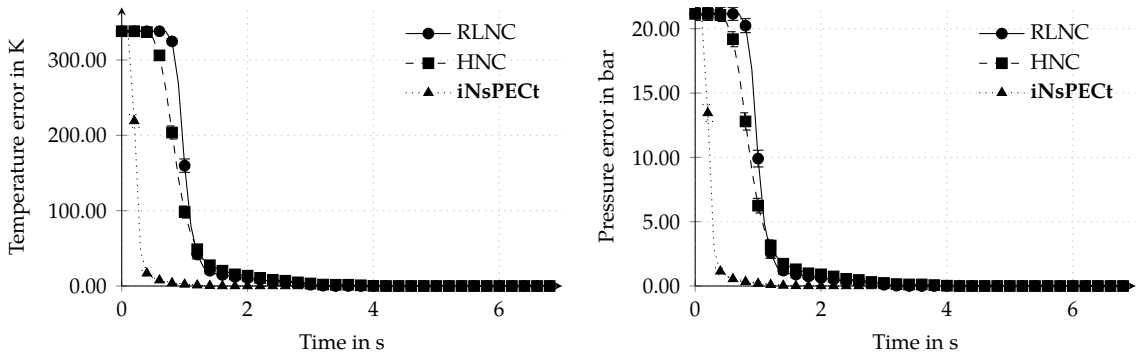
So in summary, one generation encompasses information from one particular machine i, one sensor j, and one production cycle k. Only the production cycle counter k increases over time, we thus map that counter to the generation number field in the general header format (see Figure 5.6). The origin ID must consist of both sensor number j and machine ID i to uniquely identify a cycle. Since our factory setting is a local area network where we can control IP addresses, we simply use a unique part of the IP address as machine ID and only store the sensor number in the header.

To apply prioritized network coding techniques to our industrial use case, we pre-process sensor information so that it can be divided into different priority layers, as described in Chapter 3. We apply the DCT to each production cycle's sensor information and divide its output into blocks of coefficients. Blocks with low-frequency coefficients provide an early preview of a complete sensor cycle, whereas blocks with high-frequency coefficients incrementally increase that preview's accuracy to enable more demanding fault detection techniques. We again use one injection cycle as a generation, but take the blocks' coefficient frequency as basis for prioritization. High priority layers thus contain only low frequency coefficients, which convey more information. Low priority layers, due to their cumulative nature, contain both low frequency and high frequency components.

Factory network model and evaluation

Our topology represents a typical factory layout with rows of machines in a regular grid. It is identical to the grid topology given in Figure 5.7b. But instead of having few senders and many receivers, the industrial use case requires only one sink node, n_s, that is connected to the factory's central storage and processing system. The remaining fifteen nodes represent the machines, which record sensor information from the production process. We consider two sensors for each machine, so after a production cycle, each node starts to transmit two generations in parallel, as described in Section 5.10.

Again, we use our real, pre-recorded sensor information of the injection molding process. Our sensor information stems from 25 s long production cycles that were sampled at 500 Hz rate. Analogously to Section 5.11, we split frequency components into four priority layers ($\boldsymbol{r} = (10, 10, 15, 15)$) with a generation size of $n = 50$ frequency com-



(a) Average temperature error.

(b) Average pressure error.

Figure 5.11: Approximation's average sensor error over time.

ponents to limit each data message's size to 1 kB. During each run, several production cycles are transmitted to the sink.

Analytical overhead assessment

An important consideration for any network coding scheme is message overhead. Recall from Section 5.3 that in any network coding scheme, messages consist of an encoding vector $\boldsymbol{c}^{(j)}$ and an information vector $\boldsymbol{x}^{(j)}$. Each information vector holds b symbols, so, when neglecting message overhead due to linear dependency, the required n linear combinations to decode a generation contain $n \cdot b$ symbols, which is identical to the number of symbols in the generation's message set $\boldsymbol{M}$. Consequently, message overhead (again, neglecting linear dependency) originates from the *encoding vector* only. We can thus summarize each message's overhead as the ratio between encoding vector size and information vector size, that is, n/b. In our proposed mapping of process information to generations, the generation size n is determined by the product of a production cycle's duration t, the sample rate r, and the size of each sample in symbols divided by b. Assuming no more than four symbols per sample,[10] we can thus determine the message overhead for cyclic sensor information as:

$$\text{Overhead} \;=\; \frac{4 \cdot t \cdot r}{b^2}. \tag{5.23}$$

[10] Which is in line with our system model from Chapter 2

With our concrete set of sensor information, the 25 s production cycle duration and 500 Hz sample rate, we obtain 5 % message overhead, which we consider sufficiently low. Different industrial parameters may result in larger message overhead, for instance, due to higher sample rates or longer production cycles. In this case, increasing the information vector size b should be considered while keeping in mind that this might negatively affect packet loss [71].

Simulative results

Figures 5.11a and 5.11b show the simulation results for temperature error over time and pressure error over time. The measurement starts when the first message is being transmitted. Several messages are

required before frequency components become decodable, which explains the initially very high average error that results from production cycles without any frequency components decodable at the server. It can be seen that the preview provided by the PNC scheme iNsPECt quickly gains precision. It is virtually indistinguishable from the original sensor information much earlier than RLNC can provide any information at all. HNC gains precision more quickly on average than RLNC, as well. The overhead of the HNC scheme, however, results in RLNC providing the full picture before HNC can lower the remaining error below 1 K or 1 bar. In contrast, iNsPECt achieves such a low average error approximately four times as fast as RLNC for both temperature and pressure readings. The maximum time until each production cycle was available with full precision was 3.4 s with our baseline RLNC. As a result of the principal message overhead imposed by PNC schemes, iNsPECt and HNC required up to 4.2 s and 6.0 s, respectively, until the preview reached full precision. Especially with iNsPECt, however, the residual error is extremely low ($\leq$ 3 mK and $\leq$ 2 mbar) even before RLNC finishes transmission.

Summarizing, we evaluated the impact of prioritized network coding for the smart-factory use case, using our real sensor information from the injection-molding process. Our results confirm the trends shown in Section 5.11 and suggest that iNsPECt provides significant benefits over non-prioritized RLNC, whereas HNC can only provide a coarse preview before RLNC provides the full picture.

5.13 Chapter Summary

This chapter introduced non-traditional, network-coding based network architectures. Network coding is disruptive in that it is not based on the classical store-and-forward pattern. Network coding thus breaks compatibility with existing protocols that assume routing in lower protocol layers. Often, network coding introduces high computational overhead, but it can offer throughput benefits that go beyond what is possible with store-and-forward architectures. The advent of more powerful embedded systems thus renders network coding a promising research direction for future smart factories.

In this chapter, we identified two potential showstoppers when it comes to network coding and the smart factory use case that we characterised in Chapter 2. A key requirement in our use-case is the preview functionality, which, in the case of network coding, requires early decoding of an underdetermined system of linear equations. The first showstopper is how long the decoding process takes with multiple priority layers, and the second is the question how priority layers can be selected efficiently during the encoding process.

To address the first concern, we contributed a novel decoding algorithm based on a joint representation of priority layers in a single decoder state. On top of that, we contributed a layer-selection mechanism in the encoding scheme that is designed for low decoding delay but at the same time keeps overhead from linearly dependent combi-

nations low. The evaluation results show that our decoding approach is faster and requires less memory than existing decoding approaches, and that our layer-selection scheme does provide faster delivery of prioritized information than existing approaches. Compared to the previous chapters, these contributions are more fundamental in nature: our proposed decoder provides asymptotically better performance than existing approaches, and our proposed layer-selection mechanism provides performance that is in many cases very close to the theoretical optimum. To support more applied research in this field, we also contributed an easy-to-implement mapping between network coding generations and industrial process information, and we demonstrate that this mapping only introduces tolerable message overhead.

Despite the similar industrial settings, the use of network coding, as proposed in this chapter, offers different trade-offs compared to the multi-hop protocols from Chapter 4: network coding implements redundancy over all nodes and thereby renders reliable information delivery much more likely even if multiple forwarding nodes fail during transmission. Moreover, network coding enables decoding in a strict linear manner, that is, without any ''gaps'' in the spectrum that we saw in the mechanisms from Chapters 3 and 4. Consequently, it is much easier to quantify the preview error. The mechanisms do not require the complex e-value bounds from Chapter 3.[11] On the other hand, the native support of redundant transmissions causes greater overhead in the network, which is a trade-off to consider depending on the particular industrial requirements at hand.

[11] Although the simple ''e-rest'' bound by itself is still helpful in the network coding scenario.

Given our positive results, we consider network coding a promising candidate for future industrial process control and industry automation, especially when transmission redundancy is desired.

6
Evaluating With Native Stacks: Discrete Event Protocol Emulation

6.1 Overview

This chapter is a revised and extended version of the author's publications [91] and [92, © 2019 IEEE].

Protocol evaluation is an integral part of network protocol design. Since protocol complexity increases with smart factory trends, protocols become harder to understand intuitively. It is therefore more important than ever to systematically assess protocols. Network simulations are a common choice for protocol assessment, as they offer precise control over otherwise external influences, such as wireless interference. At the same time, simulations support more detailed protocol models than analytical evaluations, and we have used simulative evaluation techniques extensively throughout chapters Chapters 3 to 5

From the perspective of experimental design, simulations constitute a middle ground between analytical protocol evaluation and testbeds. Compared to testbeds, a major drawback of simulations is that existing protocols require a separate implementation in the discrete event model. In this chapter, we propose a novel architecture to evaluate unmodified, binary protocol implementations in state-of-the-art discrete event simulators by utilizing the operating system's system call barrier, which separates user-space processes from the operating system kernel. Not requiring protocol sources is especially useful in the industry, where implementations are often proprietary. We have utilized this architecture to obtain accurate simulative results in Chapter 4.

This chapter first provides an overview of evaluation methods and existing simulation techniques before it describes our proposed architecture. Based on an exemplary implementation of our architecture for the Linux platform and the ns-3 simulator, we then demonstrate the validity and feasibility of our approach using widely used, real-world networking protocols. We show that our approach more closely resembles realistic protocol performance when compared to simulator-based protocol models. Moreover, we show that our approach performs better than existing solutions for more realistic protocol simulations.

6.2 The Case for Native Protocol Evaluation

Parallization of simulations is often used to obtain a statistically meaningful sample size much quicker than it would be possible with real-time measurements. This is especially true for wireless protocol evaluation, where wireless interference thwarts parallelization of measurements within the range of interference in test beds.

Complementing analytical and testbed-based evaluation methods, simulative approaches have proved to be a powerful tool for evaluating network protocols. The most common [74] network simulator class, discrete event simulation, combines several beneficial properties. By giving full control over otherwise external influences on simulation results, discrete event simulations allow to *reproduce* results much easier than, e. g., using testbeds. Moreover, the decoupling of simulated time and real time allows to run single simulations faster and allows multiple simulations to run in parallel.

Finally, discrete event simulators support *perfect repeatability*, which means that repeating simulations with the same parameters results in exactly the same results. Thereby, discrete event simulators allow to scrutinize protocol behavior caused by rare network constellations, such as a certain sequence of packet losses or delays.

To be executed in a simulator environment, however, protocol code may have to be re-implemented within the simulator's programming framework. Simulative evaluation results, therefore, may suffer from lacking realism due to simplistic re-implementations of network protocols [12]. In the worst case, network protocols are implemented twice and independently, once for real world deployment and once for simulative evaluation. Making the matter worse, existing protocols that are used in the industry may only have proprietary implementations, making it even harder to create a simulator based implementation. Besides the increased development and maintenance costs, possible differences in independent protocol implementations threaten to invalidate statements derived from simulation results.

Consequently, it has been a goal of the networking research community to find techniques that serve as a vessel for native protocol implementations within a network simulator [17, 52, 65, 86, 88, 125, 129]. Such a technique would improve realism by using native protocol implementations instead of simpler, simulator-based replications. Moreover, it would support existing protocols in the sense that it would neither require re-implementations of nor changes to existing protocol implementations. Over the last decade, we have seen several attempts to find such a ''vessel.'' Solutions so far, however, either impose significant restrictions on protocol implementations or outright break discrete event simulation guarantees, such as perfect repeatability.

Our proposed and novel architecture, named discRete event protocol emulatIon vesseL (gRaIL), allows to combine native protocols with discrete event simulations. For the first time, our approach allows to use virtually *any* network protocol in a state-of-the-art discrete event network simulator. gRaIL is based on loading *unchanged binary* protocol implementations through a technique called *system call wrapping*. It inspects, replaces, and re-implements relevant input and output operations exclusively based on the so-called system call barrier. This barrier usually separates applications from the host operating system, and it has convenient properties for protocol emulation. We show that gRaIL is able to accommodate native state-of-the-art routing protocols with a small impact on simulation performance and more realistic results compared to simulator-based protocol stacks.

6.3 Approaches to Protocol Evaluation

When designing or revising communication protocols, we aim to understand their behavior in different scenarios and using different protocol parameters. Approaches to evaluation employ *models* as an abstraction to achieve desirable evaluation qualities. Differences between approaches' qualities are summarized Figure 6.1. We first introduce each quality and contextualize it with related terms that are used in different fields.

Validation of models has been defined as the "substantiation that a model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model."

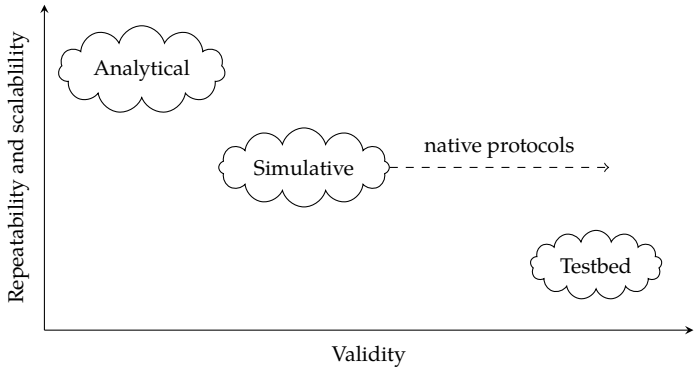


Figure 6.1: Overview of different protocol evaluation techniques. © 2019 IEEE

[113] *Accuracy* describes the model predictions' closeness to the real system, and sufficient accuracy means that the model can be used as a substitute for the real system in experiments [19]. When combining several models, a study's validity depends on every model's accuracy.

Repeatability means experimenters are able to repeat a previously conducted evaluation and observe the same overall result. It is a fundamental requirement of any experiment [40], and it has been described as a prerequisite for validity [82]. We use the term *perfect repeatability* when not only the experiment's overall result but also all observable intermediate results can be repeated exactly. *Reproducibility,* by extension, describes the ability of other researchers to reproduce the original results. Results in the networking community often suffer from lacking reproducibility that results from insufficient documentation of the evaluation methodology [74, 78]. Moreover, some evaluation approaches, such as complex testbeds, limit reproducibility due to their exposure to real-world influences – even if all relevant information is supplied.

Scalability characterizes the feasibility to evaluate a large number of network nodes, vary numerous experiment parameters, and use a large experiment area, amongst others. A method's scalability thus limits the domain for which valid results can be obtained with reasonable cost.

Next, we distinguish between *analytical* and *testbed* evaluations in terms of these qualities and argue how *simulative* evaluations have evolved as a combination thereof. Analytical evaluation – in the context of network protocol design – involves deriving a simplifying model for both the physical environment and the protocol's behavior. George Box stated that "all models are wrong, but some are useful." [14] Being an abstraction, every model is wrong when used out of context. Thus, a model should be developed for a specific purpose, and its validity should be determined with respect to that purpose [113]. If the model is valid, results reflect the protocol's expected behavior for that purpose. Perfect repeatability is achieved since analytical expressions yield the same result each time. Analytical evaluation serves well to derive asymptotic results that characterize protocol behavior for very large systems. With an inaccurate model or wrong or undocumented assumptions, results do not reflect real-world protocol behavior.

Testbed-based evaluations run protocol implementations on real-

hardware nodes. As neither hardware nor software is modeled, testbed studies often possess a high degree of accuracy. The realism comes at the cost of reduced scalability: hardware cost and effort increase with testbed size. Evaluation repeatability is not guaranteed with testbeds either, since external influences cannot be controlled. Testbed experiments are often hard to reproduce, since specific testbed conditions, such as wireless interference, cannot easily be reproduced.

Broadly summarizing, analytical models enable *repeatability, reproducibility, and scalability,* whereas testbeds often provide strong *validity.* As a compromise, researchers often use network simulators, such as ns-3 [56] or OMNeT++ [131], to evaluate their proposals. As shown in Figure 6.1, network simulations can be regarded as a middle ground between analytical protocol assessment and testbeds.

Modern network simulators usually use simplifying models for physical layer aspects of protocols, such as wireless transmissions. These models have been validated by real-world experiments for a wide range of applications [54, 79, 127]; and they allow to control environmental variables to an extent not possible using testbeds. Thereby, we can better isolate the effects of using the proposed network protocol versus alternative implementations. Simulations can be highly reproducible when all models used and their respective parameters are published along with the results.

Higher protocol layers, in contrast, are re-implemented within the simulator, closely mimicking their real-world equivalents. A protocol layer implemented within the simulator is effectively an extremely detailed model for the real-world protocol. Most notably, the protocol under evaluation is either re-implemented – and thus modeled – within the simulator environment. Or, it is prototyped in the simulator before its potential real-world implementation. In both cases, the separate implementation is an abstraction and thus has a limited domain of applicability. Validation has to be performed for each intended application of the model, and the domain of applicability must be documented explicitly to avoid user errors [113]. Such implementation differences between re-implemented protocol layers and their real-world counterparts have been identified as a cause of error in the past [12]. In contrast to physical layer aspects, models of higher protocol layers do not help control external influences and thus provide no inherent benefit. Rather, they are a prerequisite to evaluate protocols by simulation.

Therefore, it is desirable to simulate *native* protocols to reduce the amount of modeling required and thereby improve simulation validity. Using native protocols as models within simulations can currently be performed at the source level, the build or compilation step, or by means of virtualization. Each method inherently restricts applicability to certain protocols, removes simulator guarantees such as perfect repeatability, or affects scalability. Mechanisms that improve simulation realism *and* do away with major restrictions are an active area of research and may significantly contribute to objective, quantitative protocol evaluation.

6.4 Simulation and Emulation Techniques

We compare existing simulation and emulation techniques that aim to improve realism in terms of requirements imposed on the evaluated protocol: the availability of a protocol's source code, its implementation's programming language, and assumptions of how the protocol is implemented exactly. Finally, we consider whether the approach affects experiment repeatability, reproducibility, scalability, or realism.

Status quo: partial source reuse

We term the simplest form of code reuse between a protocol's real-world implementation and a simulator implementation *partial source reuse.* It resembles the traditional approach where researchers either implement a novel algorithm directly within the simulation framework's model or re-implement an existing protocol. This approach requires (a) that the protocol implementation's sources are available or yet to be written, and (b) that their (desired) programming language is compatible with the simulator language. In case of existing code, it is then possible to copy and paste chunks of source code to the network simulator implementation and execute them as part of the simulation. Several software components need porting or re-implementation to work in a discrete event simulator.

- Real-world socket APIs cannot be used in a simulator; instead, the network abstractions provided by the simulator must to be used.
- Time is different from the system time in a network simulator, so no system time queries must be made.
- Random numbers have to stem from the simulator's pseudo-random number facility exclusively.
- Concurrency is often not supported by discrete event simulators; instead the asynchronous event dispatcher of the simulator has to be used.
- Global variables may prevent spawning more than one application instance in a simulator.
- File system operations may conflict when more than one application instance is simulated.

Protocol implementation fragments can be distinguished based on whether they cause such side effects. Following compiler theory conventions, we term fragments without side effects *pure* as opposed to *impure* fragments, which cause side effects. More specifically, we denote a fragment as pure when its instructions have no directly observable effect outside of the protocol's process and no outside information is obtained by those instructions. Impure program fragments, in contrast, involve input or output operations as described in the list above, e. g., sending to or reading from a network socket. The out-of-process environment interacted with by input/output operations should be

the simulated environment, not the system environment in which the simulator is executed. Therefore, each impure fragment's execution has to be replaced when using the partial source reuse approach.

We conclude that partial source reuse can help alleviate duplicate implementation efforts. However, the approach by no means eliminates those efforts; all of the above issues have to be addressed manually. Moreover, working with existing protocol versions may be impossible due to tight restrictions on protocol implementations.

Full source reuse

Recent research [17, 41, 52, 65, 86, 125, 126] has investigated code reuse to avoid duplicate implementations. Here, we discuss approaches based on sharing a protocol implementation's entire *source code* for simulation and real-world deployment. We distinguish two different variants: employing a software compatibility layer and using alternative compilation methods. For both approaches, the sources must be available, and further restrictions on the sources apply, as described below.

Software compatibility layer Huang et al. [61] propose using a software compatibility layer that abstracts network operations and other operations that influence simulation results. Following the approach, existing software has to be modified to use abstractions instead of the operating system's native functionality. Click [88] takes a special approach: network protocols are implemented in a modular fashion in both C++ and a domain-specific router configuration language. Click protocol implementations can be deployed on real hardware or integrated with a simulator, such as ns-3 [126]. The compatibility layer's aim is to find suitable programming abstractions for the critical software components listed in Section 6.4. Protocols are implemented against this compatibility layer instead of APIs that are specific to the real world or simulation environments. The Click approach only works when developing a new protocol, as tight integration with Click is needed. Another restriction is that the compatibility layer can only support those features that *all* supported platform APIs provide, i. e., their least common denominator.

Mayer et al. [86] consider a lightweight compatibility layer for OMNeT++. Instead of compiling sources into an executable, a shared library is built and dynamically loaded into the simulator. The authors suggest to replace the network functionality with a compatibility wrapper. Thereby, network functionality, calls that query the current time, e. g., can be exchanged if the protocol is built for real-world deployment instead of simulation.

Liu et al. [79] apply the same compatibility layer approach in an even more lightweight way for a comparative study of mobile ad hoc network (MANET) routing protocols. Here, the existing routing protocol implementations' sources are modified manually. During this process, impure parts are replaced by a thin wrapper re-implementing

input/output operations with simulator equivalents if the protocol is compiled for simulation.

Alternative compilation Tazaki et al. [129] propose a refined shared library approach for the ns-3 simulator. The proposed technique, with some remaining restrictions, allows a protocol implementation's sources to run *unmodified* in the simulator. In particular, the sources run without using a compatibility layer. Instead of using a compatibility layer (which requires source code changes), calls to the operating system's standard libraries are redirected to a wrapper library. The wrapper library either passes the call to an operating system library (for most calls) or provides an alternative implementation based on simulator facilities. For example, a function call that normally returns the current system time is replaced by a wrapper that returns simulation time instead. The approach can be used for kernel space protocols in a similar fashion [65, 128]. Kim et al. [67] extend the idea to allow easy transition from simulated protocol layers to emulated layers and further to testbed protocol implementations. Ivey et al. [64] implement NVIDIA CUDA [96] bindings to show that the approach is applicable to GPU based algorithms as well.

Shared library approaches suffer, however, from several remaining conceptual restrictions: compilation to a shared library requires that the source code is available and in fact can be compiled into such a compatible library. The latter does not hold for most interpreted programming languages and even many mainstream compiled languages, such as Java or Go. In addition, the directly executed protocol must not issue system calls directly, as that would bypass the wrapper library.

Protocol emulation

A quite different approach is to run node processes or even the nodes' operating systems via virtualization and solely exchange network traffic between these real processes and the simulation. In the context of the OMNeT++ simulator, this approach was first briefly discussed in [86] and later implemented by Staub et al. [125]. The ns-3 simulator offers such emulation support with the TAP bridge module [2].

The advantage is that full code reuse is trivial, since processes run in the same environment as they would when deployed. No programming language limitations or tool-chain restrictions apply. Unfortunately, this approach does not maintain perfect repeatability, because only network operations are simulated. Processes or operating systems do not run in the simulated time domain, but in their respective system time domain. Slight time variations that result from non real-time operating systems and unpredictable system (pseudo-)random numbers further limit repeatability. Despite the limited *repeatability,* Handigol et al. [52] have successfully used protocol emulation through process reuse to *reproduce* important network experiment results.

The emulation approach generally requires careful performance isolation and resource provisioning. Otherwise, results are systematically

biased as processes compete for shared resources such as CPU time [52]. Even when these precautions are taken, the operating system scheduler and randomness-related system calls still leak uncontrolled entropy into the simulation results. Thus, network experiments with the same parameters and same random generator initialization seeds do not usually produce the same results with the emulation approach. When analyzing rare network constellations, this is a significant restriction, and generally a step back from discrete event simulation's perfect repeatability guarantees.

As a special case, simulator protocol models can be used with real networks [1]. Here, the development starts with a simulator protocol model. That protocol model is then combined with a network simulator running in real time. The simulator passes networking operations to the host system's network hardware. Carneiro et al. [18] show that this approach allows for fast prototyping in ns-3 with sufficient performance for control traffic and to some extent (forwarded) user traffic. The performance was later improved by Fontes et al. [41] by moving the data plane (which handles user traffic) out of the simulator and into either user space or kernel space. This approach can eliminate duplicate implementation efforts, but – similar to software compatibility layer approaches – requires a model implementation that is compatible with the simulator. Different to gRaIL, existing implementations cannot be used as is, and novel protocol development must be performed with techniques that are compatible to the simulator.

In summary, we can distinguish existing approaches by the ''barrier'' they choose between a native protocol and the discrete event simulation. Both the partial source-reuse approach and software compatibility layers work on the source level. The alternative compilation approach goes one step further and works on the compilation step. Finally, the protocol emulation approach virtualizes the native protocols' operating systems and thus works below the system call level, but thereby looses perfect repeatability.

6.5 *Discrete Event Process Emulation*

We argue that the ideal composition of native protocols and a discrete event simulator is to load binary protocol implementations to enhance evaluation realism. Using binaries avoids strict requirements on source code availability, programming language, language features, and so forth. At the same time, most of the environment outside of the protocol under evaluation should be as controlled as possible in order to ensure repeatability. In particular, fully virtualizing the protocol's operating system would make it difficult to control entropy.

Therefore, we argue that the most suitable barrier between protocol and simulator lies between protocols' processes and the host operating system. Modern operating systems implement exactly this kind of barrier in form of the *system call interface.* The system call interface is a natural fit for native protocol loading, as processes are generally not allowed to perform impure operations – unless they issue a system call.

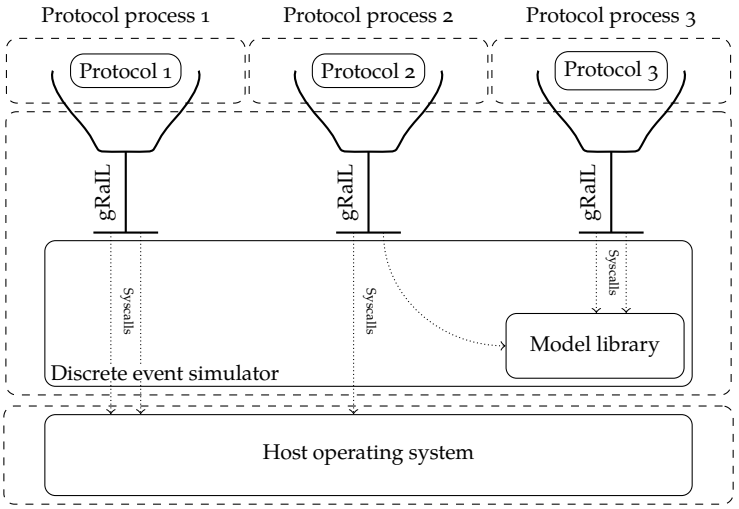


Figure 6.2: Discrete event protocol emulation architecture. © 2019 IEEE

That is, operations that cause input/output are exclusively performed via system calls. On most operating systems, the system call barrier is accessible via a kernel interface, making it feasible to implement an interception layer at this interface.

Therefore, we use the operating system's system call interface as the basis for our discrete event process emulation technique. In the following, we first give an overview of our approach's general architecture before we discuss in more detail how our approach handles system call processing. To establish the feasibility to rewrite all necessary system calls for process emulation, we derive a taxonomy that separates system calls into six categories. This categorization gives rise to a compact system architecture. We complement the feasibility discussion with a system call usage analysis based on the Linux operating system as an example platform.

Architecture overview

Our architecture's main design goals are: support for simulation validity and repeatability, interoperability with existing discrete event simulators, and feasibility to be implemented using existing operating system frameworks.

An overview of our proposed architecture is depicted in Figure 6.2, where three native protocols are used in a network simulation. Dashed lines show process boundaries. In the top part, each simulated node's emulated native protocol runs as an individual user space process. Modern discrete event simulators, e. g., ns-3 and OmNET++, run in a single process that contains all relevant models and the main event loop. In our proposed architecture, we extend the simulator process with one dedicated gRaIL-component per emulated protocol. We keep the simulators' single threaded control flow for interoperability. Each gRaIL-component processes its respective protocol's system calls and either forwards them to the host's kernel or re-implements them based on existing simulation models.

A convenient property of the system call barrier is process isolation.

Having protocols in their own operating system processes results in better protocol isolation compared to simulator-based protocol stacks. For instance, our approach shields global variables from other processes, which would be available to all protocol instances in the traditional approach. Similarly, alternative-compilation-based approaches require numerous countermeasures to isolate native protocols' memory regions from each other [129].

We take into account existing interfaces that are already implemented by operating systems to offer system call interception. In case of Unix/Posix systems, this interface is `ptrace`, previously used to implement debuggers or application firewalls. The `ptrace` framework allows to intercept and modify system calls at two points in time: first, when the monitored process issues a system call, but before the kernel executed the call. Second, after a system call was processed by the kernel, but before the monitored process sees the result.

System call processing

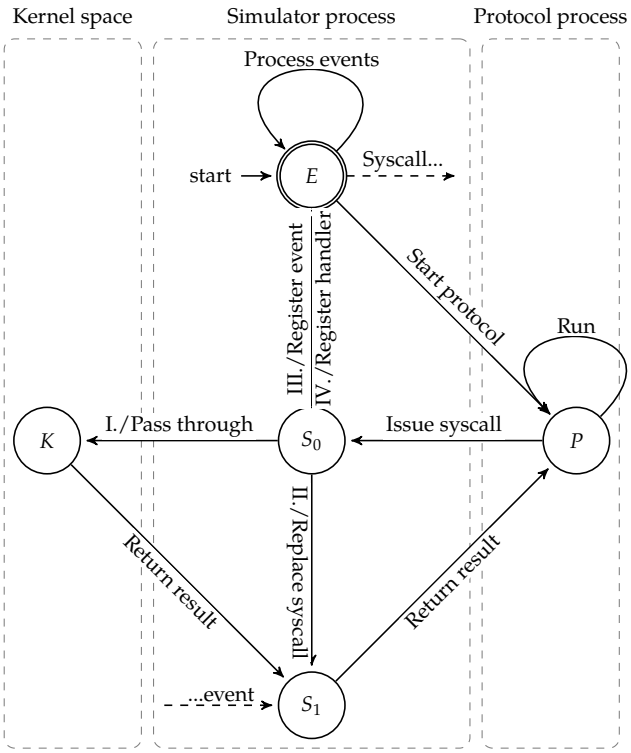


Figure 6.3: Protocol emulation's control flow as a state machine. © 2019 IEEE

gRaIL's main state machine is shown in Figure 6.3 for a single emulated process. This state machine is a direct result of the requirements posed by discrete event simulators and system call interception capabilities that we discussed in Section 6.5.

The simulator's main discrete event loop is shown as state E, where the next event is processed via the reflexive edge. We extend this main loop with states S_0 and S_1. State S_0 corresponds to a system call intercepted after it was issued, but before it was processed by the kernel; state S_1 corresponds to a system call intercepted after it was

processed by the kernel, but before its result is returned to the process. The three states arranged vertically in the middle, namely E, S_0, and S_1, constitute the simulator process. The left-hand state K signifies the kernel of the simulator's host operating system, whereas the right-hand state P is the emulated protocol process.

The control flow starts, same as without our emulation technique, with the main discrete event loop. The simulator's main loop ($E \rightarrow E$) takes the, in simulated time, soonest event from its event loop and processes it. Processing an event might influence the simulated world state, e. g., through a packet being sent. At some point, the emulated protocol is started through a simulator event ($E \rightarrow P$). This event causes the emulated protocol's process to be spawned and monitored for system calls by the host process (the simulator in our case). Thereby, the simulator yields execution to the emulated protocol's process, which continues ($P \rightarrow P$) until it issues a system call. Once a system call is issued by the emulated protocol, the native protocol's process is blocked by the kernel. The simulator is notified and enters state S_0, where it examines the system call based on its type and arguments ($P \rightarrow S_0$). Further processing depends on this examination where the system call is assigned a certain category. In any case, a result is returned eventually from state S_1.

System call categories

In order to implement system calls with the gRaIL architecture, we have to map typical system call behavior to the mechanics of discrete event simulators. The following, exhaustive list of categories determines how system calls are processed by the simulator in states S_0 and S_1. The categories were derived from three system call characteristics: (1) whether the system call affects simulation results, (2) at what point in (simulated) time the system call's result is available, and (3) whether the system call terminates the process. The resulting categories are summarized in the following list:

Cat. I: The system call has no effect whatsoever on simulation results and is passed through to the operating system kernel.

Cat. II: The system call is re-implemented in terms of what the simulator provides (e. g., a pseudo-random number). The result can be obtained at once and is returned within the same point of simulated time.

Cat. III: The system call requires re-implementation, but the result is only available at a later point in time. Thus, an event is registered with the simulator.

Cat. IV: The system call requires re-implementation, but the result is not available and depends on conditions, so a handler is registered with the simulator.

Cat. V: The system call is a hybrid of categories III and IV.

Cat. VI: The system call terminates the process.

Category I is the simplest; it encompasses, for instance, setting internal memory permissions, allocating new memory, or switching the process image. None of these system calls affects – or is affected by – the simulated environment. Therefore, these calls are not handled by the simulator, but by the operating system's kernel ($S_0 \rightarrow K$). Therefore, category I system calls need not be modified.

If a given system call potentially affects simulation results, it must be re-implemented within the simulator. In this case, we must consider that discrete event simulations are inherently asynchronous. That is, future results are entirely handled by spawning events for a future time. System calls, in contrast, are to a large extent synchronous. They halt a process's execution until a result is available and returned. One important distinction is, therefore, whether the system call is synchronous or asynchronous (i. e., blocking or non blocking).

We can further subdivide system calls based on their return behavior. Synchronous calls may return at (a) a fixed time in the future or (b) return based on external circumstances (e. g., reading from a socket until data arrives). When neither (a) nor (b) are true, we have an asynchronous system call that we list as category II. If either (a) or (b) are true, the system call falls in category III or category IV, respectively. If both (a) and (b) are true, we have a category V call.

Category II of system calls require re-implementation. For instance, the `gettimeofday` system call normally returns the current system time. But if an emulated native protocol requests the current time, the simulated time is returned instead by the protocol emulator. The simulator immediately returns the system call's result ($S_0 \rightarrow S_1$ in Figure 6.3) without further interaction with the kernel or main event loop.

Categories III and IV of system calls have in common that the system call's result is not immediately available. The third category includes system calls with a fixed return time. An example system call with a fixed return time is `nanosleep`, which makes the process sleep for a specified amount of time. In this case, an event is registered with the simulator ($S_0 \rightarrow E$). This event ($E \rightarrow S_1$) is executed after the specified amount of simulated time, e. g., the duration parameter of the `nanosleep` system call. The system call's result is then returned to the the protocol's process. In the case of `nanosleep`, the result is the simulated-time duration that the process slept.

Category IV comprises system calls that block until certain conditions become true, e. g., waiting for the arrival of a network packet on a specific socket. These system calls cannot be handled via scheduling discrete events at a fixed future point in time. Instead, they require registering callback handlers in simulated components such as sockets ($S_0 \rightarrow E$). Once the callback handler is executed, it translates the simulated environment relevant to the system call back to the process and returns the result. For example, consider a simulated socket that receives a simulated network packet that the process waits for. This packet is translated to a native operating system packet and passed back

to the process as the system call's return value (or modified argument).

A number of common system calls are hybrids of categories III and IV and thus category V. Hybrid handling is necessary whenever a system call is blocking, but it also allows to set a timeout. For instance, such a system call may wait for new information on a socket, but only up to a maximum time. In that case, a handler is registered (identically to category IV) for the condition that should result in continuing the blocked call. *In addition,* and in the same way as for category III, an event is registered for the specified timeout duration.

Finally, some system calls, which we list as category VI, are used to terminate the process in case of an error condition or when the process successfully finishes execution. In both cases, the wrapper is no longer needed: all registered events and registered handlers of the process are canceled and control is given back to the discrete event simulator. Before that, however, the call is passed through to the kernel to have the native protocol's process terminate. Therefore, category VI is a special case of category I in the depicted state machine where S_1 removes all remaining state.

In summary, category I catches all pure system calls, whereas category VI is a special case that marks the end of process emulation. Categories II to V result directly from mapping possibly synchronous system calls to inherently asynchronous discrete event simulators.

System call usage analysis

While implementing the handling of system calls is a manual process, not all software will use all available system calls. To understand development effort and feasibility, we analyze system call usage on Linux/amd64 as an example platform. Specifically, we analyze system call usage for Debian ''Stretch,'' a common Linux-based server operating system.

Debian's kernel, which is Linux 4.9, specifies a total of 404 system calls, 314 of which are supported by amd64. A much smaller number of system calls is actually used by typical network protocol implementations. To quantify this argument, we analyze all available *network category* software packages.

We inspect system-call symbols that we find in protocol *binaries* and recursively all *dynamic libraries* that these binaries require. Theoretically, it is possible that a program dynamically calculates system calls not found as symbols in its binary or dependent libraries. But in practice, there is no reason to obfuscate sources or binaries in such a way. On the other hand, it is rare that all functions of a library are used, especially when a large dynamic library is loaded. Consequently, not all system calls are actually issued by protocols that depend on libraries, rendering the analysis result a plausible upper bound.

Debian's network category consists of 1941 packages. Of those, only 1790 packages contain non-GUI software, which is suitable for simulation. Of these non-GUI packages, 877 packages contain binaries to analyze.

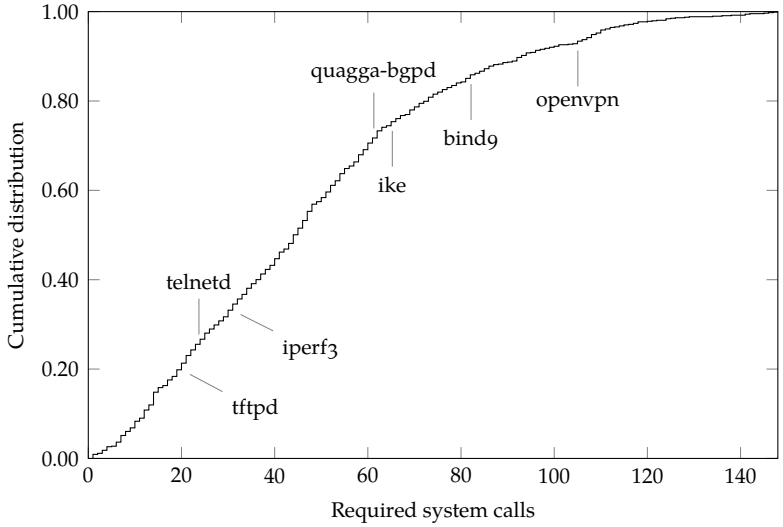


Figure 6.4: Cumulative distribution of required system calls per network package. © 2019 IEEE

The number of system calls in each of the packages' binaries is shown as a distribution plot in Figure 6.4: the x-axis shows packages sorted by the number of required system calls and the y-axis shows the cumulative distribution. Common network protocol implementations are marked. The maximum number of required system calls for an individual package is 148. However, only a total of 228 distinct system calls was found over all packages, i. e., only 56.4 % of the Linux 4.9 system calls are used at all. Therefore, we conclude that implementing system call interception with compatibility for all network software packages is feasible, especially when considering that their implementation is a one-time effort.

6.6 Linux-based Implementation

To demonstrate gRaIL's viability, we describe our example implementation on the Linux/amd64 platform. We describe in detail how each system call category identified in Section 6.5 is implemented. We omit category VI, as it is just a small variation of category I. For ease of reading, we use exemplary representatives of each category during the descriptions.

Category I: passthrough

Since category I is the simplest to implement, we use it to describe the conceptual handling of system calls in detail. First, a protocol is started by a discrete event. As an example, assume this protocol later issues a `brk` system call.

When initially starting the protocol, the simulator first executes `fork` to create a second process, `execve` to switch that process's image to the native protocol binary, and uses the `ptrace` framework to configure system call *tracing*.

Next, the (parent) simulator process waits for the native protocol to issue a system call and subsequently handles that system call, as shown

Algorithm 6: Conceptual handling of a category I system call. © 2019 IEEE

```
 1: function SYSC_HANDLER(pid)
 2:     WAIT_FOR_SYSCALL(pid)
 3:     syscall_id ← GET_CHILD_REG(pid, rax)
 4:     switch syscall_id do
 5:         case SYS_mprotect:
 6:         ...
 7:         case SYS_brk:
 8:           WAIT_FOR_SYSCALL(pid)
 9:           REGISTER_EVENT(SYS_HANDLER, 100 ns)
10:           break
11:         case SYS_gettimeofday:
12:         case SYS_getrandom:
13:         ...
14:     end switch
15: end function
```

in Algorithm 6: line 2 is the call to a function that waits for a system call via the `ptrace` framework. Specifically, the `PTRACE_SYSCALL` command is used to execute the traced process until it issues a system call. The simulator process waits for such an event using `waitpid` and checks that the resulting status code is indicative for a system call event. In order to identify an issued system call and find the category it belongs to, we use the system call's numeric identifier. On Linux/amd64, this identifier is located in the protocol process's `rax` register, which is read in line 3.

The simulator now determines the correct category for the system call's numeric identifier using the switch statement in lines 4 to 14. Each ''case''-expression matches a specific system call using its Linux-specific symbolic identifier. Here, lines 5 to 7 match our category I system call `brk`. Lines 11 onward match different system call categories.

A category I system call is not modified in any way. Therefore, the simulator continues execution of the process in line 8 until the system call's result is available from the kernel. Finally, in line 9, the system call handler re-registers itself to the discrete event simulation's main event loop. Here, we add a delay of 100 ns simulated time to avoid so-called ''poll loops,'' which could exist in native protocols. The emulated protocol's process remains halted after the function finishes. When the function reaches line 2, the protocol's execution is continued.

Our description so far only considered a single protocol process being executed. Multiple protocol processes in simulation are support as follows: as only one process is executed at the same time, at most one process is being waited for at any given point in time. To simulate multiple processes, execution between processes is switched whenever a system call is issued. That way, parallel protocol simulation is linearized analogously to executing ordinary system processes on a single-core CPU system – except that the switch-over point is not based on resource usage, but on processes' system call execution.

Category II: substitution

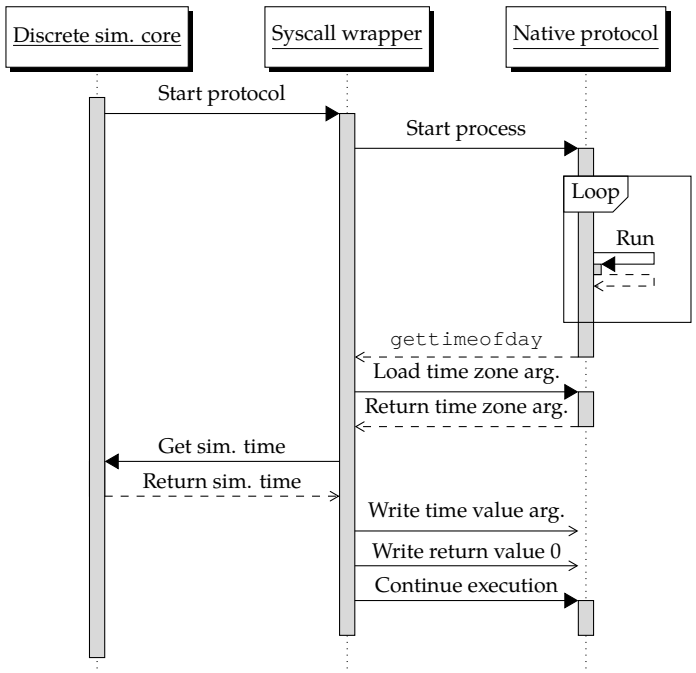


Figure 6.5: A category II system call.

The processing of a category II system call is shown as a sequence diagram in Figure 6.5. Recall that category II comprises all system calls that return immediately but need to be modified in order to maintain repeatability. Common examples are getting the current time or obtaining a random number. The strategy to handle these system calls is to replace their return value with values controlled by the simulator. Discrete event simulators already implement the necessary state management and methods, so gRaIL only needs to implement proxy methods for the corresponding system calls.

In Figure 6.5 we discuss the implementation using `gettimeofday` as a representative for this system call category shown. The wrapper intercepts and inspects the system call and classifies it as a category II call, i. e., no involvement of kernel code is required to determine the call's result itself. The handler procedure for this particular system call is executed and, as a first step, loads the system call's arguments into the simulator's memory. In case of `gettimeofday`, the first argument is the a time-value data structure into which the current time is to be written; the second argument is a data structure that indicates the time zone in which the date should be returned. On Linux/amd64, arguments to system calls are passed as CPU registers to the kernel. These CPU registers are mapped to a special memory area that can be read from and written to via `ptrace`. In the case of `gettimeofday`, both arguments are passed as pointers, so the CPU registers only contain pointers to emulated protocol specific memory regions. As a second step, the system call handler thus copies this memory to a local variable in the simulator process to inspect the time zone argument. Next, the handler obtains the current *simulation* time from the simulator core, converts it according to the time zone argument, and copies it into the

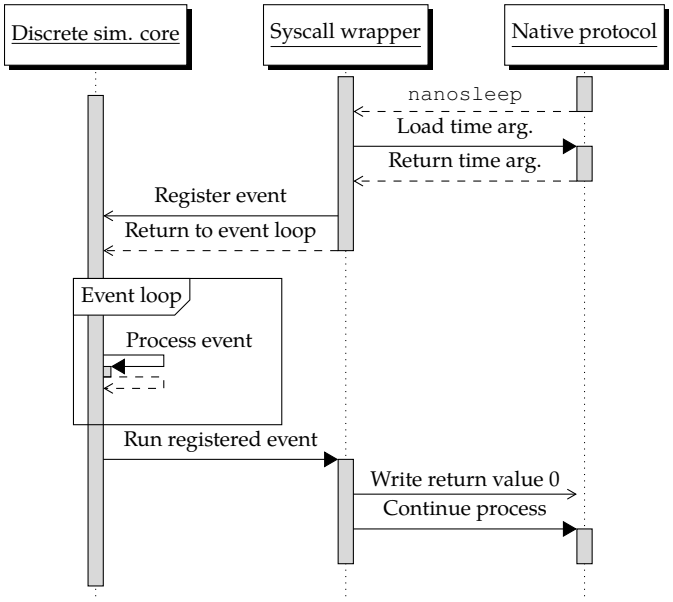


Figure 6.6: A category III system call (omitting protocol process initialization). © 2019 IEEE

native protocol's memory location that is given by the system call's first argument. Finally, the return code 0, which indicates successful completion of `gettimeofday`, is written to the process's `rax` register, which is used for system call return values.

Category III: timer-triggered

In contrast to category II, system calls in category III do not return immediately. As their return is after a fixed time period, however, these system calls can conveniently be handled by the discrete event simulator's event queue. We describe the process using `nanosleep` as a representative system call.

Figure 6.6 shows an example sequence diagram for the `nanosleep` system call, assuming initial protocol startup has already taken place. The `nanosleep` system call has only one parameter: a data structure that holds the time the process wishes to sleep. The system call wrapper loads this argument from the process's memory and parses the time value. Next, it registers an event with the simulator that is triggered after the relative time value argument duration; the simulator then continues executing its main event loop. The native protocol's process remains sleeping during this period. After the specified duration of simulation time, the registered event is triggered: the system call wrapper writes the return value and continues execution of the native protocol's process.

Category IV: context-triggered

Figure 6.7 describes handling of the category IV system call `recv`, which is used to read network packet data from a socket. The `recv` call has four parameters: the socket number, a pointer to a buffer for the received packet's payload, an indicator of the buffer's maximum length, and an optional option field. We assume that when the system

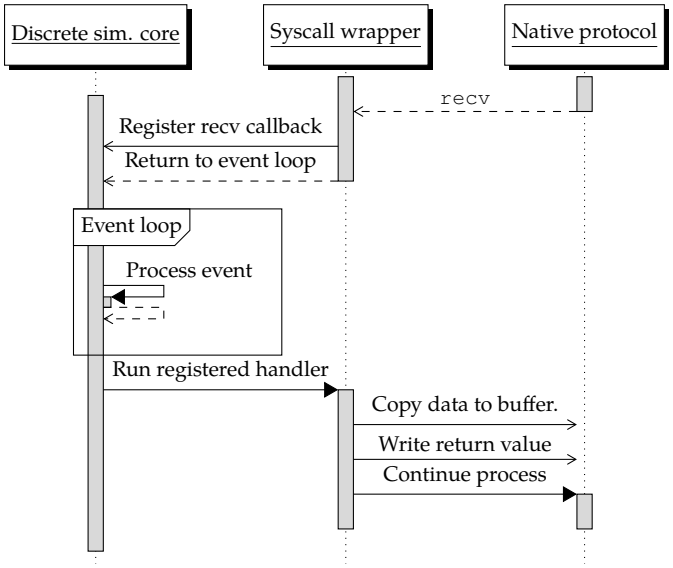


Figure 6.7: A category IV system call (omitting protocol process initialization). © 2019 IEEE

call `recv` is issued, a valid socket to receive data from was already created in the simulation. Despite having a pointer argument for the buffer, it is not necessary to load the associated memory region. The wrapper only writes to this region when receiving packets.

Category IV system calls block for an unspecified duration. In our case, that is until a packet is received by the socket. Therefore, a *callback handler* is registered with the simulator. As soon as a registered handler is executed, it copies the data received to the memory region in the protocol's process, honoring the buffer's maximum size. Before the process is continued, the number of bytes copied is written to the return-value register of the protocol process.

Category V: hybrid

The system call `select` is a typical representative of category V, which are hybrids of category III and category IV: `select` allows a process to register callbacks for socket-related events, e. g., receiving a new packet. `select`, however, also allows to specify a timeout duration after which the call should return even if no registered event occurred. If the registered timeout event is executed prior to the handler, it un-registers the handler. Conversely, the handler un-registers the event when it is executed first.

Moreover, `select` allows to conditionally block on *several* network sockets' events and specify a timeout. We handle such behavior in our architecture by having the first handler or timeout event that is run un-register all remaining handlers and events. Thereby, the protocol's process is halted until all obsolete handlers and events are un-registered.

6.7 gRaIL Evaluation

We evaluate gRaIL using a Linux-based implementation for ns-3 [56], which supports more than 55 exemplary system calls. In addition, we implement the functionality of several kernel sub-systems that can be accessed through system calls.

The evaluation focuses on *model verification and validation,* i. e., whether gRaIL-based evaluations impede results, and *simulator performance,* i. e., whether gRaIL is feasible for practical simulation scenarios. As application scenarios, we use two examples for commonly used networking software:

1. an *ad-hoc networking scenario* using the open link state routing daemon Olsrd [99], and
2. a *wired networking scenario* using Iperf [63], a tool for performing network throughput measurements.

For each scenario, we describe the implemented simulation set-up, and the verification, validation, and performance results.

Olsrd implements open link state routing (OLSR) [28], a MANET routing protocol, as a user space daemon that modifies the operating system's routing tables via kernel interfaces. The Olsrd software is widely used, e. g., connecting over 400 nodes in the Berlin area as part of the Freifunk network [44]. A version of OLSR is implemented as part of the ns-3 distribution. However, ns-3's implementation does not implement all of the native distribution's improvements. Therefore, OLSR is a good candidate for evaluation, as it allows to compare differences between a common protocol's re-implementation for discrete-event network simulators and real-world protocol code.

Iperf 3 is a common network performance analysis tool, best known for its TCP-based throughput tests. Iperf 3 implements a control channel to negotiate benchmark parameters and a separate TCP or UDP connection to perform the actual benchmarking, in the following jointly referred to as the "Iperf protocol." Iperf serves to evaluate gRaIL's suitability for wired networks. It also serves as an example for a TCP-based protocol in contrast to Olsrd's UDP-based message exchanges.

We highlight that our proof-of-concept implementation of gRaIL consists of fewer than 3000 non-empty, non-comment lines of code, as reported by `cloc`. Yet, it runs native Iperf and Olsrd protocols in simulations. Iperf, in comparison, counts more than 8000 lines of code, whereas Olsrd consists of over 30 000 lines of code. Even the simplified OLSR model in the ns-3 simulator counts more than 5000 lines of code.

We use ns-3 version 3.28, settings are summarized in Table 6.1. YANS Wifi model [76] is configured with IEEE 802.11g media access control (MAC) and 2.4 GHz physical layer. We consider an obstructed line of sight and account for multi-path propagation and large-scale path loss by employing Rayleigh and log-distance propagation loss models, one superimposed on another [54]. For wired simulations, links are modeled as full-duplex point-to-point links. To obtain independent samples for wired simulations, we employ a (pseudo) ran-

Wireless setting	Value
Transient phase, measure duration	150 s, 300 s
UDP traffic generation	OnOffApplication (CBR mode)
Area per node	15 m to 20 m radius disc
802.11g channel	Channel 1, 20Mhz ch. width (*)
Phy. rate adaption mechanism	ErpOfdmRate54Mbps (const. rate)
Path loss exponent	$\gamma = 3.0$ (*)
Reference path loss at 1 m	40.1 dBm (Friis at 2.4 GHz)
TX-power	16.02 dBm (*)
Antenna gain, cable atten.	0 (*)
Error rate model	NistErrorRateModel (*)
Wired setting	**Value**
Simulation duration	100 s
Point to point error model	1 bit error per s
RTT	40 ms (=10 ms link delay)
Iperf TCP-flow duration	10 s (Iperf 3 default)

(*) ns-3.28 default setting

Table 6.1: Simulation settings.

domized receive error model with an error rate of one bit error per second. Simulations run on a system with an Intel Core i7 CPU (four cores, eight threads, 2.7 GHz clock), 24 GiB main memory, and Debian ''Stretch'' (kernel version 4.9.0).

We repeat each simulation at least ten times, using independent sub-streams of ns-3's MRG32k3a pseudo-random number generator (PRNG) [75]. Unless otherwise specified, all data points show the sample mean, and error bars indicate 95 % confidence intervals (using the non-parametric, bias-corrected and accelerated bootstrap method [30]). Error bars may not be visible when the confidence is very high.

Ad-hoc networking scenario (Olsrd)

Simulation set-up Figure 6.8 shows the simulated topology: we use a random disc topology of n nodes that communicate over wireless links. In addition, a client node C generates generates and sends UDP packets towards the server node S, which acts as a sink, at a constant bit rate (CBR). As indicated by the solid lines in Figure 6.8, C and S are connected via point-to-point links to their respective inner disc nodes.

Client and server nodes are solely simulated by ns-3, thus, all CBR user traffic is generated *within* ns-3. Depending on the simulated scenario, the inner disc nodes' OLSR implementation is either provided by ns-3, gRaIL, or ns-3's TAP bridge. Figure 6.9 gives an overview of the components and relevant data flows in each scenario: most notably, only the TAP bridge involves routing user traffic through Linux Container (LXC) virtualization and updates to routing tables *outside* of the ns-3 simulator. In the case of gRaIL and TAP bridge, OLSR control traffic is generated *outside* of the simulation.

The setup requires multiple OLSR features: nodes C and S each

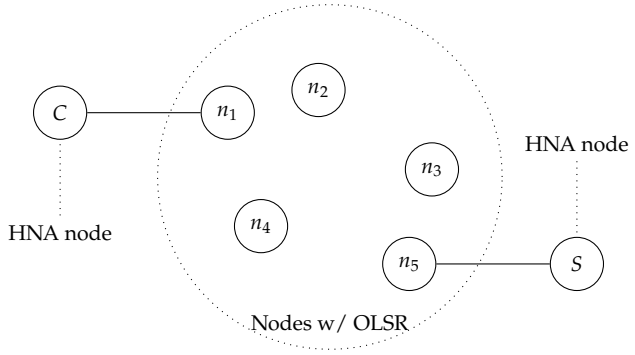


Figure 6.8: Random disc wireless topology for Olsrd simulations (here: $n = 5$). © 2019 IEEE

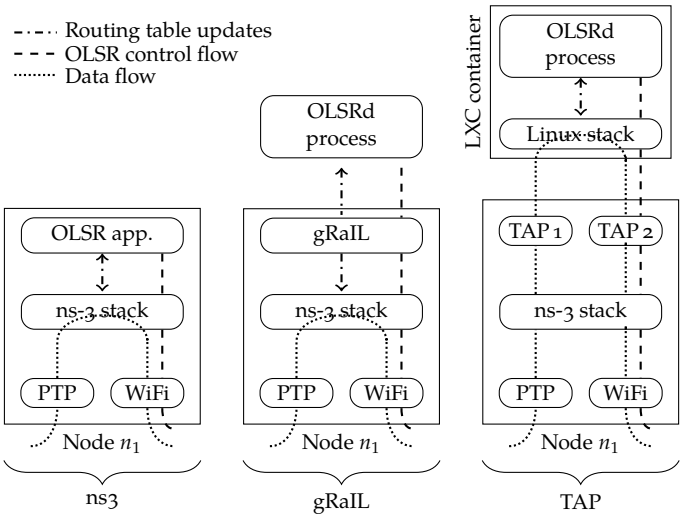


Figure 6.9: Components of node n_1 in ns-3, gRaIL, and TAP scenario. © 2019 IEEE

reside in separate IP subnets, which are propagated throughout the whole network via OLSR host native interface messages; and the disc nodes find shortest multi-hop paths in the wireless topology and set routing tables accordingly. The simulated small-scale and larger setups facilitate verification and validation, as well as checking of model accuracy and simulation performance.

Verification of perfect repeatability To verify our expectation that gRaIL maintains perfect simulation repeatability, we generated random simulation topologies and executed each one repeatedly using the *same* random seed. For each simulation, we recorded both user traffic and OLSR control traffic in *pcap* files, which contain each frame's content and headers with precise timestamps in simulated time. In addition, we configured ns-3 to enrich pcap files with *radiotap* headers. These headers show, among other information, the received signal strength indicator (RSSI), i. e., the signal strength with which each frame was received by the simulated wireless network adapter. The RSSI is highly variable for each frame due to the noise introduced by the Rayleigh fast fading model. Therefore, observed RSSI in the simulation is a strong indicator for the presence or absence of potential input or output leaks in gRaIL's implementation.

We executed a total of 10 simulation scenarios with random disc topologies (n=30). We repeated each simulation three times, using

the same configuration and initialization of ns-3's random number generator. Next, we checked whether the pcap files were *byte identical* for simulation scenarios that used the same random seed and set-up and, conversely, verified that they differed for scenarios with different random generator initializations.

100 % of the captured traffic was identical when the simulated scenario was identically configured, and 100 % of the generated pcap files differed when the scenario differed in any respect. Note that the pcap traces contain precise timestamps, which means each frame was sent and received at the exact same time, with the exact same RSSI, and fully identical content, including all headers. We consider these results strong evidence that gRaIL is able to maintain perfect repeatability.

Validation against an emulated testbed The goal of validation is a model that accurately predicts the performance of its real-world counterpart [19]. gRaIL loads unmodified, real-world protocol binaries into a discrete event simulator. The combination of gRaIL with a protocol binary can therefore be considered a (very detailed) model of the protocol.

Ideally, we only want to validate the accuracy of gRaIL in isolation, not (at the same time) the accuracy of, e. g., ns-3's path loss and fading models, bit error model, MAC model, and so forth. The TAP bridge component of ns-3 [2] allows us to observe the behavior of the real-world Olsrd protocol implementation, running in LXC virtualization. As discussed in Section 6.4, ns-3's TAP bridge is an example for an emulated testbed [53]. Using such an emulated testbed, we can compare gRaIL-Olsrd to an Olsrd instance that runs in real time and sets actual Linux routing table entries, but uses the same simulated topology, wireless model, and shares much of the network stack (compare Figure 6.9).

We focus on small, random disc topologies with only five inner-disc nodes (and thus five virtual machines). This setup accommodates a common weakness of emulation-based approaches: increased system load due to large topologies could impede model accuracy. Over 200 simulations with one node per 20 m radius disc density, we observed an average packet delivery ratio (PDR) of 82.8 % with gRaIL and 82.1 % with the emulated test bed. The similar, cumulative distribution of the observed PDRs can be seen in Figure 6.10. Beside the similar PDRs, Figure 6.10 shows that more than half of the topologies allow for very good routes, which result in PDRs close to 100 %, and about 10 % of topologies are challenging with PDRs in the range 0 % to 10 %. Thus, for further statistical analysis, we have to take into account that the observed PDR is highly correlated with the PRNG initialization, which determines the random topology.

To validate the gRaIL-based simulation against the emulated testbed, we consider the similarity of the PDR for each generated topology using *paired* tests on identical topologies. The mean difference per topology in PDR was 0.8 percentage points. Next, we quantify the pairwise similarity between the models with a statistical *equivalence* test. The null hypothesis is that the gRaIL model for Olsrd yields different re-

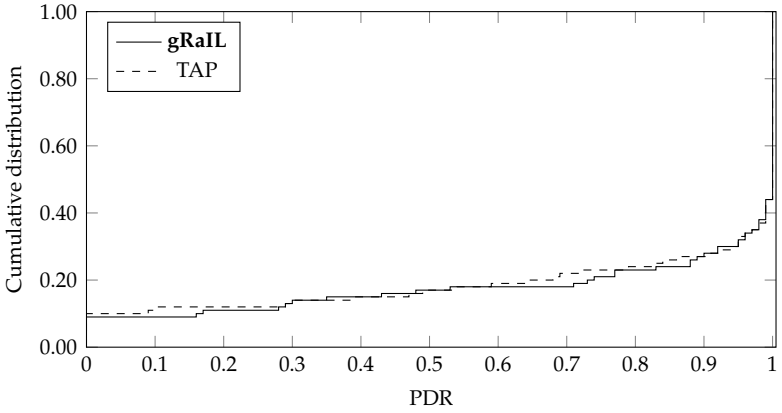


Figure 6.10: PDR distribution for OLSR random disc simulations. © 2019 IEEE

sults in terms of PDR than the emulation-based testbed. We first note that the PDR distributions, according to a Shapiro-Wilk normality test, are not normally distributed ($p = 1.63 \times 10^{-22} \ll 0.05$). Consequently, we cannot use Wellek's equivalence test. Instead, we use the non-parametric, regression-based two one-sided test (TOST) [111], a bootstrap-based statistical equivalence test specifically designed for model validation. This test compares pairs of a prediction (gRaIL's PDR) to an observation (TAP's PDR): the test rejected the null hypothesis of differences between model and observation (at $\alpha = 0.05$, margins of similarity $\epsilon_0 = 0.05$ and $\epsilon_1 = 25\,\%$). Similarly, a robust TOST equivalence test (Schuirmann's TOST test [118], using Yuen's robust t-test [140]) rejected the null hypothesis of different models as well ($\alpha = 0.05$, paired test variant, margin of similarity $\epsilon = 0.05$). We conclude that gRaIL and TAP bridge models for loading the real-world OLSR implementation are equivalent for the simulated scenarios.

Model accuracy in a larger scenario Next, we look at scalability aspects and consider larger and more dense topologies. Here, we also compare ns-3's internal OLSR model to the TAP and gRaIL approaches.

The original OLSR protocol uses a simple hop-count metric to determine shortest paths. The Olsrd daemon, in contrast, uses the more fine grained expected transmission count (ETX) [31] metric per default. This ETX metric accounts for varying quality levels of links and is used in the next protocol version's proposed standard as well [27]. The simulator ns-3 only implements the older, hop-count-based draft standard [28]. In the following, we compare ns-3's OLSR model to a similarly configured Olsrd daemon that is emulated with the proposed gRaIL approach. Additionally, we show results for gRaIL-loaded Olsrd with its default configuration, which has the ETX protocol extensions enabled, and compare results to an emulation testbed, as well.

Figure 6.11 shows PDR results for larger random disc topologies. Here, the x-axis gives the varying number of nodes in a random disc topology. To avoid artifacts from too dense topologies (where the PDR would be close to 100 % most of the time), we increase the disc area proportionally to the number of nodes, effectively keeping the node density in the disc constant (with one node per 15 m radius disc

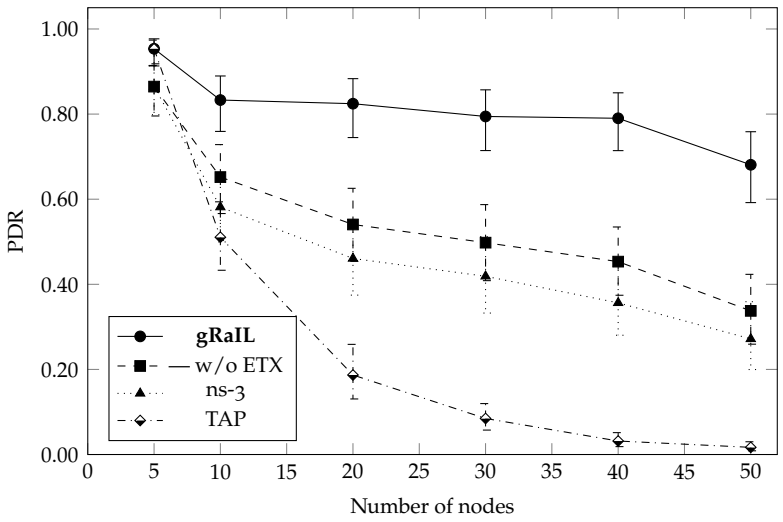


Figure 6.11: OLSR model performance in terms of PDR. Varying number of nodes in random disc topology (with constant node density). © 2019 IEEE

area). Since OLSR only determines routing table choices for the inner-disc nodes, the PDR-relevant user traffic does not pass the system call barrier. Consequently, gRaIL's model performance advantage in terms of PDR over ns-3 in Figure 6.11 must result exclusively from routing table choices, making an unfair bias of the gRaIL architecture over ns-3 unlikely.

The results show a significant PDR advantage (8.9 to 43.4 percentage points) of the gRaIL-modeled real-world OLSR implementation (default configuration with ETX), when compared to ns-3's internal OLSR model. Different from gRaIL, the TAP bridge approach routes the actual user traffic through virtual machines, which means that the virtual machines are responsible for acknowledging MAC frames. Traffic thus has to pass from the simulator process to the forwarding node's LXC container and back; the same is true for MAC layer acknowledgments. The TAP bridge model, being a real-time approach, is susceptible to delays resulting from such overhead, and the observed PDR worsens faster than with the other models in the larger topologies. At $n = 10$ nodes, the PDR is already 7 percentage points worse than the ns-3 model, at $n = 50$, the remaining PDR is only 1.7 % and thus 25.5 percentage points worse than the ns-3 model. In terms of PDR, the native OLSR implementation on average outperforms the ns-3 OLSR model consistently with 6.6 percentage points – even when it is running without ETX extensions. This confirms previous results from Bikov et al. [12], who observed similar model performance differences between ns-3's OLSR model and the real-world implementation in earlier versions of ns-3 and Olsrd.

Our findings confirm that differences between native protocol implementations and simulator-based protocol models can severely impact the results' applicability to the real world, be it because of unnoticed errors in the model or non-standard extensions that are used in native implementations.

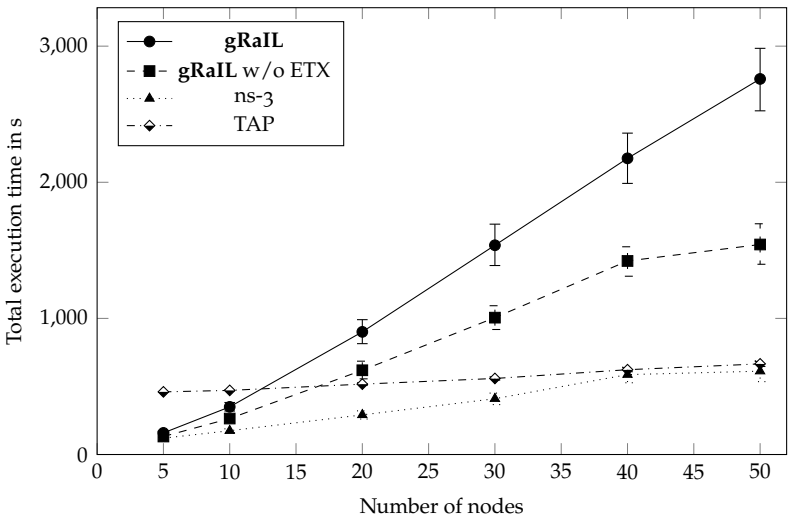


Figure 6.12: OLSR simulator performance in terms of execution time. Varying number of nodes in random disc topology (with constant node density). © 2019 IEEE

Simulator performance Next, we quantify the simulation execution time using gRaIL compared to that of vanilla ns-3 simulations and the TAP bridge approach to evaluate scalability and performance. The results are shown in Figure 6.12, where the x-axis is chosen identically to Figure 6.11, but the y-axis shows total simulation execution time. The TAP bridge approach provides almost constant runtime with respect to the number of simulated nodes: if the simulation executes faster than system time, it is throttled by the TAP bridge module. Should, however, the simulation fail to keep up with the real-time emulated OLSR software, it causes packet drops due to timeouts. The TAP bridge thus suffers in model performance with increasing system load, whereas the other approaches suffer in simulation execution time, which can be seen by comparing Figure 6.12 to Figure 6.11. From the remaining two OLSR models, ns-3 provides the best simulator performance. In comparison, gRaIL-based simulations took (with the RFC 3626 conformant, native OLSR), from 10.2 % to 152.0 % more time to finish. With the ETX calculations enabled in the OLSR protocol implementation, the overhead increases by 20.1 % to 78.8 % compared to the RFC configuration. We note that the overhead is, in part, due to the higher PDR, which causes more load in the simulator for forwarding the CBR user traffic.

Finally, we consider both simulator and protocol-process memory consumption using `smem`, which reports proportional set size, a metric that takes into account both per-process memory consumption and shared memory. Over 10 repetitions, gRaIL's and ns-3's simulator processes required on average 57.6 MB and 57.7 MB, respectively. For 50 nodes, gRaIL uses 60.0 MB main memory and ns-3 66.7 MB. In addition, gRaIL's OLSR processes each required 347.4 kB and 352.1 kB for $n = 5$ and $n = 50$. That means for 50 nodes, gRaIL simulations require about 11.0 MB more memory than ns-3, which is much less than typical simulation systems' available main memory.

Wired networking scenario (Iperf 3)

Simulation set-up We now show results for executing the TCP-based Iperf protocol with gRaIL and compare its model accuracy and simulator performance to the alternative compilation approach direct code execution (DCE) [129].[1] We choose DCE for comparison here as the best-known alternative compilation approach for ns-3. The simulated topology is one Iperf server node connected to one Iperf client node via an intermediate router node R. Figure 6.13 shows each simulated scenario's components and traffic flows for node n_c.

Note that all Iperf traffic is generated outside of the ns-3 model stack, but forwarded over router node R's inside the simulation.

Having the latest Iperf protocol version, Iperf 3, run as part of a discrete event simulation with DCE required several preparatory steps. First, we had to find third-party patched sources, as the original sources used unsupported system library functions and contained a poll loop that results in an infinite loop with DCE. Next, the C sources had to be compiled using specific compiler options to assure DCE-compatibility of the resulting binary: (1) the binary has to be a dynamic ELF binary; (2) the binary has to contain all symbols in the symbol table (and not only symbols that are actually used); (3) the binary must consist of position independent code; and (4) the binary must not invoke ''fortified'' library functions, which include additional range checks to avoid out of bounds accesses, since those are not supported by DCE.

Using gRaIL, none of these steps is required and it is possible to simply use the official Iperf binary without any modification.[2] gRaIL is capable of loading the modified Iperf version used by DCE as well, which we use for comparability.

[1] DCE supports several kernel-space protocol stacks. For comparability, we have configured DCE to use the ns-3 stack.

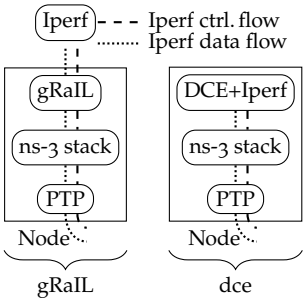


Figure 6.13: Components and control flow: DCE vs gRaIL. © 2019 IEEE

[2] The poll loop in the original Iperf implementation, while not resulting in an infinite loop, can negatively affect the performance of gRaIL.

Verification and validation against an existing model We first perform the same comparison as in Section 6.7 on a sample of 10 simulation results with varying channel capacity in 1 Mbit/s to 30 Mbit/s: given an identical scenario and identical PRNG initialization, all pcap-recorded traffic was identical in every way for gRaIL. The same was not true for the DCE model, which we attribute to an unimplemented system library function that reports, among other statistics, processor utilization for the executing process. As DCE leaks this system-provided information into the simulation, it produces different traffic traces with each simulation, regardless of PRNG seeds.

Apart from the latter not being perfectly repeatable, we could not determine significantly different results between gRaIL and DCE when executing the Iperf simulations. Figure 6.14 gives the cumulative distribution of observed throughput. We performed a total of 50 simulations with channel capacity varying from 1 Mbit/s to 30 Mbit/s. The observed throughput, except header overhead, closely matches the channel capacity for both models, which is visible by the almost-matching steps in the cumulative distribution. In addition, both the regression-based TOST equivalence test [111] (with $\alpha = 0.05, \epsilon_0 = \pm 25\,\%, \epsilon_1 = \pm 25\,\%$) and the pairwise, robust TOST (with $\epsilon = \pm 0.3$ Mbit/s) rejected

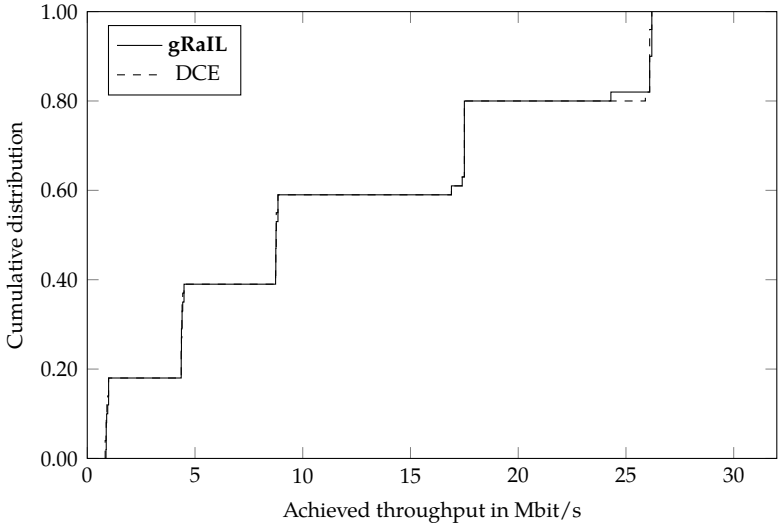


Figure 6.14: Cumulative distribution of observed throughput. © 2019 IEEE

the null hypothesis of different model behavior in terms of throughput (at $\alpha = 0.05$).

Simulator performance Figure 6.15 shows total execution time as a function of channel capacity. The figure gives a lower bound based on a much simpler ns-3 simulation using ns-3's `BulkSendApplication`, which only models an equal-duration TCP transmission without the control channel, exchange of results, or actual payload. Regardless of whether DCE or gRaIL is used, both native Iperf protocols fully saturate the channel, which means there is no unfair bias between implementations. With larger channel capacity, Iperf sends more data over its TCP socket, and therefore, both protocol client and server require more time processing.

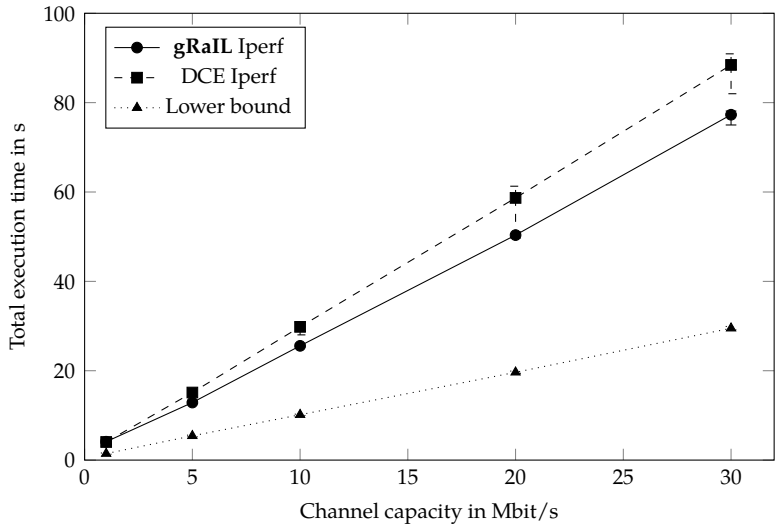


Figure 6.15: Total simulation execution time. © 2019 IEEE

Surprisingly, gRaIL consistently outperforms DCE despite the more intricate use of the system barrier as opposed to simpler function calls within a single process; at 30 Mbit/s, gRaIL's mean performance ad-

vantage over DCE is 12.6 %, and at least 38.2 % of gRaIL's simulator execution time (only 33.4 % for DCE) can be attributed to ns-3 packet processing.

Evaluation summary

Our evaluation demonstrates that the gRaIL approach to utilize the system call barrier enables simulating native, binary protocols with sufficient performance for most applications. Our results show that the performance is lower than having a dedicated protocol model in the simulator, but at the same time, it demonstrates the pitfalls of such a duplicate model: divergent protocol behavior (in this case: worse route choices). The comparison of gRaIL to the alternative-compilation-based approach DCE shows: gRaIL performs better in an Iperf-based TCP simulation. At the same time, gRaIL makes native protocols much easier to use by not requiring modifications to sources, alternative compilation, or even availability of sources.

6.8 Chapter Summary

Discrete event network simulations are a powerful tool to evaluate network protocols. A long-standing shortcoming, however, is their dependency on separate protocol models that duplicate implementation efforts and may even impede simulation validity. In this chapter, we proposed gRaIL, a novel architecture to combine *native* protocol implementations with modern discrete event simulators. Our design is based on exploiting the operating system's system call barrier to provide isolation between protocol instances and distinguish pure from impure operations. Different to existing approaches, gRaIL can execute *binary* protocol implementations and requires no modifications to or even availability of the protocol implementation's sources.

We instantiated the proposed architecture for the Linux/amd64 platform and demonstrated that it is capable of running full fledged network protocols, such as OLSR or Iperf, with sufficient accuracy and performance. We used that gRaIL implementation to improve the simulation results in Chapter 4 compared to the original publication [93];[3] we believe, though, that the proposed architecture is helpful in a much larger context and constitutes an important step towards more realistic protocol evaluation. This is especially the case with existing – possibly proprietary – protocols used in the industry.

Our implementation is available on Github as an ns-3 module for download, experimentation, and extension.[4]

[3] In the original TANDEM publication, topology information came from a custom extension of the ns-3 OLSR model, which, however, resulted in notably less accurate routes.

[4] `https://github.com/ns3grail/grail`

7
Conclusion

MANUFACTURING is an important industry sector, and industrial production is a driving motor for productivity and job creation in developed countries. In this context, smart factories not only seek to further improve productivity and flexibility, but are an answer to the challenge of the relocation of industrial production into low-wage countries. Smart factories, encompassing fully sensorized and decentralized systems, require reliable and efficient wireless protocols for challenging environments.

This work made several contributions to two integral components of industrial protocol development: protocol design and protocol evaluation. After identifying important properties of industrial protocols, we proposed several protocol mechanisms in Chapter 3 to Chapter 5, whereas Chapter 6 regarded protocol evaluation.

At the source, we showed that compression and prioritization based on the discrete cosine transform (DCT) enables a fast and approximate preview of information from the production process – with precise error bounds. Moving prioritization tasks into forwarding nodes, we showed how larger factories can benefit from such mechanisms with improved coverage, fairness, and reliability. We also solved two main challenges that disallow the utilization of network coding for industrial use cases: first, we proposed a specialized decoding technique for prioritized, network-coded information, and second, we contributed an efficient encoding scheme to minimize delay until a preview of process information is available.

Last, this dissertation examined the evaluation of industrial protocols and identified that state of the art techniques to simulate existing, possibly proprietary protocols cannot retain important properties of discrete event simulations, such as perfect repeatability. To address this shortcoming, we contributed a novel simulation technique that chooses the system-call level as the barrier between native, binary protocols and the discrete event simulation. We validated our evaluation technique on the Linux platform for existing protocols and showed that it provides sufficient performance for typical protocol evaluation applications, and we used our technique for the evaluation of protocol designs that we proposed earlier in this work.

We consider the contributions of this dissertation as important steps towards smart factories in a changing and developing industrial landscape. Although motivated by the requirements of manufacturing industry, the utility of the contributed protocol mechanisms and evaluation techniques is not necessarily limited to industrial settings. Compression and prioritized transmission of sensor information are similarly important to home automation or autonomous vehicles, and the contributions to network coding are well suited to scenarios with multiple receivers or sinks, which is common in multimedia streaming. Likewise, our proposed protocol evaluation technique simplifies or enables the simulation of existing protocol implementations, whether they are industry related or not.

Bibliography

References

[1] ns-3 developers. *Emulation Overview — Model Library*. July 20, 2018. URL: https://web.archive.org/web/20180720085120/https://www.nsnam.org/docs/release/3.28/models/html/emulation-overview.html (visited on 07/20/2018).

[2] ns-3 developers. *Tap NetDevice — Model Library*. July 17, 2018. URL: https://web.archive.org/web/20180717151257/https://www.nsnam.org/docs/release/3.28/models/html/tap.html (visited on 07/17/2018).

[3] Szymon Acedanski, Supratim Deb, Muriel Médard, and Ralf Koetter. "How Good Is Random Linear Coding Based Distributed Networked Storage". In: *Workshop on Network Coding, Theory and Applications*. 2005.

[4] A. R. Agrawal, I. O. Pandelidis, and M. Pecht. "Injection-Molding Process Control—A Review". In: *Polymer Engineering & Science* (1987). DOI: 10.1002/pen.760271802.

[5] R. Ahlswede, Ning Cai, S. Y. R. Li, and R. W. Yeung. "Network Information Flow". In: *IEEE Transactions on Information Theory* (July 2000). DOI: 10.1109/18.850663.

[6] Nasir Ahmed, T Natarajan, and Kamisetty R Rao. "Discrete Cosine Transform". In: *Computers, IEEE Transactions on* (1974).

[7] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. "Wireless Sensor Networks: A Survey". In: *Computer networks* (2002).

[8] Ian F. Akyildiz, Xudong Wang, and Weilin Wang. "Wireless Mesh Networks: A Survey". In: *Comput. Netw. ISDN Syst.* (Mar. 2005). DOI: 10.1016/j.comnet.2004.12.001.

[9] J. Alakuijala and Z. Szabadka. *Brotli Compressed Data Format*. RFC 7932 (Informational). Internet Engineering Task Force, July 2016.

[10] Steven C. Althoen and Renate McLaughlin. "Gauss-Jordan Reduction: A Brief History". In: *The American Mathematical Monthly* (1987). DOI: 10.2307/2322413. JSTOR: 2322413.

[11] Dimitri P. Bertsekas, Robert G. Gallager, and Pierre Humblet. *Data Networks*. Prentice-Hall International New Jersey, 1992.

[12] Evgeni Bikov and Pavel Boyko. "Direct Execution of OLSR MANET Routing Daemon in Ns-3". In: *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*. SIMUTools '11. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011.

[13] Sanjit Biswas and Robert Morris. "ExOR: Opportunistic Multi-Hop Routing for Wireless Networks". In: *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM 2005. ACM, 2005. DOI: 10.1145/1080091.1080108.

[14] G. E. P. Box. "Robustness in the Strategy of Scientific Model Building". In: *Robustness in Statistics*. Ed. by ROBERT L. Launer and GRAHAM N. Wilkinson. Academic Press, Jan. 1, 1979. DOI: 10.1016/B978-0-12-438150-6.50018-2.

[15] Malte Brettel, Niklas Friederichsen, Michael Keller, and Marius Rosenberg. "How Virtualization, Decentralization and Network Building Change the Manufacturing Landscape: An Industry 4.0 Perspective". In: (2014).

[16] M. Burrows and D. J. Wheeler. *A Block-Sorting Lossless Data Compression Algorithm*. 1994.

[17] Daniel Camara et al. "DCE: Test the Real Code of Your Protocols and Applications over Simulated Networks". In: *Communications Magazine, IEEE* (2014).

[18] Gustavo Carneiro, Helder Fontes, and Manuel Ricardo. "Fast Prototyping of Network Protocols through Ns-3 Simulation Model Reuse". In: *Simulation Modelling Practice and Theory* (Oct. 2011). DOI: 10.1016/j.simpat.2011.06.002.

[19] J. S. Carson. "Model Verification and Validation". In: *Proceedings of the Winter Simulation Conference*. Proceedings of the Winter Simulation Conference. Dec. 2002. DOI: 10.1109/WSC.2002.1172868.

[20] Szymon Chachulski, Michael Jennings, Sachin Katti, and Dina Katabi. "MORE: A Network Coding Approach to Opportunistic Routing". In: (2006).

[21] Phuc Chau, Seongyeon Kim, Yongwoo Lee, and Jitae Shin. "Hierarchical Random Linear Network Coding for Multicast Scalable Video Streaming". In: *Asia-Pacific Signal and Information Processing Association, 2014 Annual Summit and Conference (APSIPA)*. IEEE, 2014.

[22] Wen-Chin Chen, Manh-Hung Nguyen, Wen-Hsin Chiu, Te-Ning Chen, and Pei-Hao Tai. "Optimization of the Plastic Injection Molding Process Using the Taguchi Method, RSM, and Hybrid GA-PSO". In: *Int J Adv Manuf Technol* (Apr. 1, 2016). DOI: 10.1007/s00170-015-7683-0.

[23] Philip A. Chou, Yunnan Wu, and Kamal Jain. "Practical Network Coding". In: (2003).

[24] Delphine Christin, Parag S. Mogre, and Matthias Hollick. "Survey on Wireless Sensor Network Technologies for Industrial Automation: The Security and Quality of Service Perspectives". In: *Future Internet* (Apr. 8, 2010). DOI: 10.3390/fi2020096.

[25] X. Chu and Y. Jiang. "Random Linear Network Coding for Peer-to-Peer Applications". In: *IEEE Network* (July 2010). DOI: 10.1109/MNET.2010.5510916.

[26] J. Claridge and I. Chatzigeorgiou. "Probability of Partially Decoding Network-Coded Messages". In: *IEEE Communications Letters* (2017). DOI: 10.1109/LCOMM.2017.2704110.

[27] T. Clausen, C. Dearlove, P. Jacquet, and U. Herberg. *The Optimized Link State Routing Protocol Version 2*. RFC 7181 (Proposed Standard). Internet Engineering Task Force, Apr. 2014.

[28] T. Clausen and P. Jacquet. *Optimized Link State Routing Protocol (OLSR)*. RFC 3626 (Experimental). Internet Engineering Task Force, Oct. 2003.

[29] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag New York, Inc., 2002.

[30] Anthony Christopher Davison and David Victor Hinkley. *Bootstrap Methods and Their Application*. Cambridge university press, 1997.

[31] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. "A High-Throughput Path Metric for Multi-Hop Wireless Routing". In: *Wirel. Netw.* (July 2005). DOI: 10.1007/s11276-005-1766-z.

[32] Supratim Deb et al. "Network Coding for Wireless Applications: A Brief Tutorial". In: IWWAN, 2005.

[33] Peter J. Denning. "ACM President's Letter: Performance Analysis: Experimental Computer Science as Its Best". In: *Communications of the ACM* (1981).

[34] Richard Draves, Jitendra Padhye, and Brian Zill. "Comparison of Routing Metrics for Static Multi-Hop Wireless Networks". In: *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '04. ACM, 2004. DOI: 10.1145/1015467.1015483.

[35] Huseyin M. Ertunc, Kenneth A. Loparo, and Hasan Ocak. "Tool Wear Condition Monitoring in Drilling Operations Using Hidden Markov Models (HMMs)". In: *International Journal of Machine Tools and Manufacture* (July 1, 2001). DOI: 10.1016/S0890-6955(00)00112-7.

[36] M. Esmaeilzadeh, P. Sadeghi, and N. Aboutorab. "Random Linear Network Coding for Wireless Layered Video Broadcast: General Design Methods for Adaptive Feedback-Free Transmission". In: *IEEE Transactions on Communications* (Feb. 2017). DOI: 10.1109/TCOMM.2016.2630062.

[37] European Comission. "COMMISSION RECOMMENDATION of 6 May 2003 Concerning the Definition of Micro, Small and Medium-Sized Enterprises". In: *Official Journal of the European Union* (2003).

[38] European Comission. *Factories of the Future PPP: Towards Competitive EU Manufacturing*. 2018.

[39] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi. "In-Network Aggregation Techniques for Wireless Sensor Networks: A Survey". In: *IEEE Wireless Communications* (Apr. 2007). DOI: 10.1109/MWC.2007.358967.

[40] Dror G. Feitelson. "Experimental Computer Science: The Need for a Cultural Change". In: *Internet version: http://www. cs. huji. ac. il/~ feit/papers/exp05. pdf* (2006).

[41] Helder Fontes, Tiago Cardoso, and Manuel Ricardo. "Improving Ns-3 Emulation Performance for Fast Prototyping of Network Protocols". In: *Proceedings of the Workshop on Ns-3*. WNS3 '16. ACM, 2016. DOI: 10.1145/2915371.2915374.

[42] C. Fragouli. "Network Coding: Beyond Throughput Benefits". In: *Proceedings of the IEEE* (Mar. 2011). DOI: 10.1109/JPROC.2010.2093470.

[43] Christina Fragouli, Jean-Yves Le Boudec, and Jörg Widmer. "Network Coding: An Instant Primer". In: *ACM SIGCOMM Computer Communication Review* (2006).

[44] *Freifunk Project*. Dec. 6, 2017. URL: https://web.archive.org/web/20171206125316/https://freifunk.net/en/.

[45] R. Gallager and D. van Voorhis. "Optimal Source Codes for Geometrically Distributed Integer Alphabets (Corresp.)" In: *IEEE Transactions on Information Theory* (Mar. 1975). DOI: 10.1109/TIT.1975.1055357.

[46] Björn Gernert and Lars Wolf. "Poster: PotatoScanner: Using a Field Sprayer As Mobile DTWSN Node". In: *Proceedings of the 12th Workshop on Challenged Networks*. CHANTS '17. ACM, 2017. DOI: 10.1145/3124087.3124102.

[47] C. Gkantsidis and P. R. Rodriguez. "Network Coding for Large Scale Content Distribution". In: *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Mar. 2005. DOI: 10.1109/INFCOM.2005.1498511.

[48] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2013.

[49] Grand View Research. *Injection Molded Plastics Market Size Report By Raw Material (PP, ABS, HDPE, PS), By Application (Packaging, Consumables & Electronics, Automotive, Construction, Medical), And Segment Forecasts, 2018 - 2025*. Feb. 2018.

[50] P. Gupta and P. R. Kumar. "The Capacity of Wireless Networks". In: *IEEE Transactions on Information Theory* (Mar. 2000). DOI: 10.1109/18.825799.

[51] E. L. Hahne. "Round-Robin Scheduling for Max-Min Fairness in Data Networks". In: *IEEE Journal on Selected Areas in Communications* (Sept. 1991). DOI: 10.1109/49.103550.

[52] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. "Reproducible Network Experiments Using Container-Based Emulation". In: *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*. CoNEXT '12. ACM, 2012. DOI: 10.1145/2413176.2413206.

[53] Furqan Haq and Thomas Kunz. "Simulation vs. Emulation: Evaluating Mobile Ad Hoc Network Routing Protocols". In: *Proceedings of the International Workshop on Wireless Ad-Hoc Networks (IWWAN 2005)*. 2005.

[54] Homayoun Hashemi. "The Indoor Radio Propagation Channel". In: *Proceedings of the IEEE* (1993).

[55] H. M. Hashemian and Wendell C. Bean. "State-of-the-Art Predictive Maintenance Techniques". In: *IEEE Transactions on Instrumentation and Measurement* (Oct. 2011). DOI: 10.1109/TIM.2009.2036347.

[56] Thomas R. Henderson, Mathieu Lacage, George F. Riley, C. Dowell, and J. B. Kopena. "Network Simulations with the Ns-3 Simulator". In: *SIGCOMM demonstration* (2008).

[57] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros. "The Benefits of Coding over Routing in a Randomized Setting". In: *IEEE International Symposium on Information Theory, 2003. Proceedings*. IEEE International Symposium on Information Theory, 2003. Proceedings. June 2003. DOI: 10.1109/ISIT.2003.1228459.

[58] Tracey Ho et al. "A Random Linear Network Coding Approach to Multicast". In: *Information Theory, IEEE Transactions on* (2006).

[59] Ming-Shyan Huang. "Cavity Pressure Based Grey Prediction of the Filling-to-Packing Switchover Point for Injection Molding". In: *Journal of Materials Processing Technology* (Mar. 23, 2007). DOI: 10.1016/j.jmatprotec.2006.10.037.

[60] Qingqing Huang, Baoping Tang, Lei Deng, and Jiaxu Wang. "A Divide-and-Compress Lossless Compression Scheme for Bearing Vibration Signals in Wireless Sensor Networks". In: *Measurement* (May 1, 2015). DOI: 10.1016/j.measurement.2015.02.017.

[61] X. W. Huang, R. Sharma, and S. Keshav. "The ENTRAPID Protocol Development Environment". In: *IEEE INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings*. IEEE INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. Mar. 1999. DOI: 10.1109/INFCOM.1999.751666.

[62] Xiao Long Huang and Brahim Bensaou. "On Max-Min Fairness and Scheduling in Wireless Ad-Hoc Networks: Analytical Framework and Implementation". In: *Proceedings of the 2Nd ACM International Symposium on Mobile Ad Hoc Networking & Computing* (Long Beach, CA, USA). MobiHoc '01. ACM, 2001. DOI: 10.1145/501445.501447.

[63] *Iperf3 Project*. Dec. 2, 2017. URL: http://web.archive.org/web/20171202103558/http://software.es.net/iperf/.

[64] J. Ivey, G. Riley, B. Swenson, and M. Loper. "Designing and Enabling Simulation of Real-World GPU Network Applications with Ns-3 and DCE". In: *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). Sept. 2016. DOI: 10.1109/MASCOTS.2016.12.

[65] S. Jansen and A. McGregor. "Simulation with Real World Network Stacks". In: *Proceedings of the Winter Simulation Conference, 2005.* Proceedings of the Winter Simulation Conference, 2005. Dec. 2005. DOI: 10.1109/WSC.2005.1574539.

[66] Jamal N. Al-Karaki and Ahmed E. Kamal. "Routing Techniques in Wireless Sensor Networks: A Survey". In: *Wireless Communications, IEEE* (Dec. 2004).

[67] Young-Hwan Kim et al. "Enabling Iterative Development and Reproducible Evaluation of Network Protocols". In: *Computer Networks* (Apr. 2014). DOI: 10.1016/j.bjp.2014.01.006.

[68] Naoto Kimura and Shahram Latifi. "A Survey on Data Compression in Wireless Sensor Networks". In: *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference On*. IEEE, 2005.

[69] Jonathan Gana Kolo, S. Anandan Shanmugam, David Wee Gin Lim, Li-Minn Ang, and Kah Phooi Seng. "An Adaptive Lossless Data Compression Scheme for Wireless Sensor Networks". In: *Journal of Sensors* (2012). DOI: 10.1155/2012/539638.

[70] S. Kopparty, S. V. Krishnamurthy, M. Faloutsos, and S. K. Tripathi. "Split TCP for Mobile Ad Hoc Networks". In: *IEEE Global Telecommunications Conference, 2002. GLOBECOM '02*. IEEE Global Telecommunications Conference, 2002. GLOBE-

COM '02. Nov. 2002. DOI: 10.1109/GLOCOM.2002.1188057.

[71] Jari Korhonen and Ye Wang. "Effect of Packet Size on Loss Rate and Delay in Wireless Links". In: *Wireless Communications and Networking Conference, 2005 IEEE*. IEEE, 2005.

[72] Dimitrios Koutsonikolas, Y. Charlie Hu, and Chih-Chun Wang. "Pacifier: High-Throughput, Reliable Multicast without"Crying Babies"in Wireless Mesh Networks". In: *INFOCOM 2009, IEEE*. IEEE, 2009.

[73] Ulf Kulau, Sebastian Schildt, Stephan Rottmann, Björn Gernert, and Lars Wolf. "Demo: PotatoNet – Robust Outdoor Testbed for WSNs: Experiment Like on Your Desk. Outside." In: *Proceedings of the 10th ACM MobiCom Workshop on Challenged Networks*. CHANTS '15. ACM, 2015. DOI: 10.1145/2799371.2799374.

[74] Stuart Kurkowski, Tracy Camp, and Michael Colagrosso. "MANET Simulation Studies: The Incredibles". In: *ACM SIGMOBILE Mobile Computing and Communications Review* (2005).

[75] Pierre L'Ecuyer, Richard Simard, E. Jack Chen, and W. David Kelton. "An Object-Oriented Random-Number Package with Many Long Streams and Substreams". In: *Operations Research* (Dec. 2002). DOI: 10.1287/opre.50.6.1073.358.

[76] Mathieu Lacage and Thomas R. Henderson. "Yet Another Network Simulator". In: *Proceeding from the 2006 Workshop on Ns-2: The IP Network Simulator*. ACM, 2006.

[77] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. "Industry 4.0". In: *Business & Information Systems Engineering* (Aug. 2014). DOI: 10.1007/s12599-014-0334-4.

[78] Zohaib Latif, Kashif Sharif, Maria K. Alvi, and Fan Li. "Simulation Standardization: Current State and Cross-Platform System for Network Simulators". In: *Mobile Ad-Hoc and Sensor Networks*. Ed. by Liehuang Zhu and Sheng Zhong. Springer Singapore, 2018. DOI: 10.1007/978-981-10-8890-2_38.

[79] J. Liu et al. "Simulation Validation Using Direct Execution of Wireless Ad-Hoc Routing Protocols". In: *18th Workshop on Parallel and Distributed Simulation, 2004. PADS 2004.* 18th Workshop on Parallel and Distributed Simulation, 2004. PADS 2004. May 2004. DOI: 10.1109/PADS.2004.1301280.

[80] Thomas Löhl, Marc Weidlich, and Stefanie Wolf. "The Injection Molding Machine as Measuring Instrument". In: *Kunststoffe international* (Apr. 2016).

[81] Dominik Lucke, Carmen Constantinescu, and Engelbert Westkämper. "Smart Factory - A Step towards the Next Generation of Manufacturing". In: *Manufacturing Systems and Technologies for the New Frontier*. Springer, London, 2008. DOI: 10.1007/978-1-84800-267-8_23.

[82] Gunnar Lucko and Eddy M. Rojas. "Research Validation: Challenges and Opportunities in the Construction Domain". In: *Journal of Construction Engineering and Management* (Jan. 2010). DOI: 10.1061/(ASCE)CO.1943-7862.0000025.

[83] Enrico Magli, Mea Wang, Pascal Frossard, and Athina Markopoulou. "Network Coding Meets Multimedia: A Review". In: *Multimedia, IEEE Transactions on* (2013).

[84] F. Marcelloni and M. Vecchio. "A Simple Algorithm for Data Compression in Wireless Sensor Networks". In: *IEEE Communications Letters* (June 2008). DOI: 10.1109/LCOMM.2008.080300.

[85] Francesco Marcelloni and Massimo Vecchio. "An Efficient Lossless Compression Algorithm for Tiny Nodes of Monitoring Wireless Sensor Networks". In: *Comput J* (Nov. 1, 2009). DOI: 10.1093/comjnl/bxp035.

[86] Christoph P. Mayer and Thomas Gamer. *Integrating Real World Applications into OMNeT++*. 2008.

[88] Robert Morris, Eddie Kohler, John Jannotti, and M. Frans Kaashoek. "The Click Modular Router". In: *ACM Transactions on Computer Systems* (2000).

[95] K. Nguyen, T. Nguyen, and S. c Cheung. "Peer-to-Peer Streaming with Hierarchical Network Coding". In: *2007 IEEE International Conference on Multimedia and Expo*. 2007 IEEE International Conference on Multimedia and Expo. July 2007. DOI: 10.1109/ICME.2007.4284670.

[96] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. "Scalable Parallel Programming with CUDA". In: *Queue* (Mar. 2008). DOI: 10.1145/1365490.1365500.

[97] Hasan Oktem, Tuncay Erzurumlu, and Ibrahim Uzman. "Application of Taguchi Optimization Technique in Determining Plastic Injection Molding Process Parameters for a Thin-Shell Part". In: *Materials & Design* (2007). DOI: 10.1016/j.matdes.2005.12.013.

[98] *OLSR.Org*. 2016. URL: http://www.olsr.org/mediawiki/index.php/Main_Page (visited on 06/26/2016).

[99] *OLSRd Project*. Oct. 14, 2017. URL: https://web.archive.org/web/20171014120237/http://www.olsr.org/.

[100] B. Ozcelik and T. Erzurumlu. "Comparison of the Warpage Optimization in the Plastic Injection Molding Using ANOVA, Neural Network Model and Genetic Algorithm". In: *Journal of Materials Processing Technology* (Feb. 1, 2006). DOI: 10.1016/j.jmatprotec.2005.04.120.

[101] C. Perkins, E. Belding-Royer, and S. Das. *Ad hoc On-Demand Distance Vector (AODV) Routing*. RFC 3561 (Experimental). Internet Engineering Task Force, July 2003.

[102] Supachai Phaiboon. "Space Diversity Path Loss in a Modern Factory at Frequency of 2.4 GHz". In: *WSEAS Transactions on Communications* (2014).

[103] Plastics Europe. *Press Release: The European Plastics Industry Continues Its Stable Trend as a Continuation of Its Recovery*. 2016. URL: https://www.plasticseurope.org/en/newsroom/press-releases/european-plastics-industry-continues-its-stable-trend-continuation-its-recovery (visited on 05/09/2018).

[104] PREVIEW Consortium. *Deliverable 2.1: Data Acquisition Requirements*. Mar. 31, 2015.

[105] PREVIEW Consortium. *Deliverable 3.1: System Architecture and Requirements Specification*. Apr. 28, 2016.

[106] PREVIEW Consortium. *Preview Project EU | Predictive System to Recommend Injection Mould Setup with Process Optimisation in Wireless Sensor Networks*. Oct. 2, 2017. URL: https://web.archive.org/web/20171002023841/http://www.preview-project.eu/.

[107] Theodore S. Rappaport and Clare D. McGillem. "UHF Fading in Factories". In: *Selected Areas in Communications, IEEE Journal on* (1989).

[108] Priyanka Rawat, Kamal Deep Singh, Hakima Chaouchi, and Jean Marie Bonnin. "Wireless Sensor Networks: A Survey on Recent Developments and Potential Synergies". In: *J Supercomput* (Apr. 1, 2014). DOI: 10.1007/s11227-013-1021-9.

[109] M. A. Razzaque, Chris Bleakley, and Simon Dobson. "Compression in Wireless Sensor Networks: A Survey and Comparative Evaluation". In: *ACM Trans. Sen. Netw.* (Dec. 2013). DOI: 10.1145/2528948.

[110] RJG Inc. *Why In-Mold Sensors?* Jan. 8, 2019. URL: https://web.archive.org/web/20190108161702/https://www.rjginc.com/technology/sensors (visited on 01/08/2019).

[111] A. P. Robinson, R. A. Duursma, and J. D. Marshall. "A Regression-Based Equivalence Test for Model Validation: Shifting the Burden of Proof". In: *Tree Physiology* (July 1, 2005). DOI: 10.1093/treephys/25.7.903.

[112] Subhasis Saha. "Image Compression—from DCT to Wavelets: A Review". In: *Crossroads* (Mar. 2000). DOI: 10.1145/331624.331630.

[113] Robert G. Sargent. "Verification and Validation of Simulation Models". In: *Journal of simulation* (2013).

[116] Marie Schaeffer. "Distributed Prioritization in Network Coding". Humboldt-Universität zu Berlin, Nov. 13, 2017.

[117] Marie Schaeffer. *Studienprojekt: Hierarchical Network Coding - Priorisierung*. Mar. 9, 2017.

[118] Donald J. Schuirmann. "A Comparison of the Two One-Sided Tests Procedure and the Power Approach for Assessing the Equivalence of Average Bioavailability". In: *Journal of Pharmacokinetics and Biopharmaceutics* (Dec. 1, 1987). DOI: 10.1007/BF01068419.

[119] J.P. Sebastia, J. Alberola Lluch, J.R. Lajara Vizcaino, and J. Santiso Bellon. "Vibration Detector Based on GMR Sensors". In: *IEEE Transactions on Instrumentation and Measurement* (Mar. 2009). DOI: 10.1109/TIM.2008.2005073.

[120] Changyu Shen, Lixia Wang, and Qian Li. "Optimization of Injection Molding Process Parameters Using Combination of Artificial Neural Network and Genetic Algorithm Method". In: *Journal of Materials Processing Technology* (Mar. 23, 2007). DOI: 10.1016/j.jmatprotec.2006.10.036.

[121] Shenglan Huang, Michele Sanna, Ebroul Izquierdo, and Pengwei Hao. "Optimized Scalable Video Transmission over P2P Network with Hierarchical Network Coding". In: ICIP. 2014.

[122] B. Shrader and N. M. Jones. "Systematic Wireless Network Coding". In: *MILCOM 2009 - 2009 IEEE Military Communications Conference*. MILCOM 2009 - 2009 IEEE Military Communications Conference. Oct. 2009. DOI: 10.1109/MILCOM.2009.5380081.

[124] Hagen Sparka. *Compressing Noisy Physical Sensor Data with Two-Model-Compression*. 2017.

[125] Thomas Staub, Reto Gantenbein, and Torsten Braun. "VirtualMesh: An Emulation Framework for Wireless Mesh and Ad Hoc Networks in OMNeT++". In: *Simulation* (2010).

[126] P. Lalith Suresh and Ruben Merz. "Ns-3-Click: Click Modular Router Integration for Ns-3". In: *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*. SIMUTools '11. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011.

[127] Emmeric Tanghe et al. "The Industrial Indoor Channel: Large-Scale and Temporal Fading at 900, 2400, and 5200 MHz". In: *IEEE Transactions on Wireless Communications* (July 2008). DOI: 10.1109/TWC.2008.070143.

[128] Hajime Tazaki, Frédéric Urbani, and Thierry Turletti. "DCE Cradle: Simulate Network Protocols with Real Stacks for Better Realism". In: *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*. SimuTools '13. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013.

[129] Hajime Tazaki et al. "Direct Code Execution: Revisiting Library OS Architecture for Reproducible Network Experiments". In: *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*. CoNEXT '13. ACM, 2013. DOI: 10.1145/2535372.2535374.

[130] A. Tellaeche and R. Arana. "Rapid Data Acquisition System for Complex Algorithm Testing in Plastic Molding Industry". In: *Proceedings of World Academy of Science, Engineering and Technology*. World Academy of Science, Engineering and Technology (WASET), 2013.

[131] András Varga and Rudolf Hornig. "An Overview of the OMNeT++ Simulation Environment". In: *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*. Simutools '08. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[132] Dejan Vukobratović and Vladimir Stanković. "Unequal Error Protection Random Linear Coding for Multimedia Communications". In: *Multimedia Signal Processing (MMSP), 2010 IEEE International Workshop On*. IEEE, 2010.

[133] Gregory K. Wallace. "The JPEG Still Picture Compression Standard". In: *Communications of the ACM* (1991).

[134] D. Wang, Q. Zhang, and J. Liu. "Partial Network Coding: Theory and Application for Continuous Sensor Data Collection". In: *14th IEEE International Workshop on Quality of Service*. 14th IEEE International Workshop on Quality of Service. June 2006. DOI: 10.1109/IWQOS.2006.250455.

[135] M. Wang and B. Li. "Lava: A Reality Check of Network Coding in Peer-to-Peer Live Streaming". In: *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*. IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications. May 2007. DOI: 10.1109/INFCOM.2007.130.

[136] A. Willig, K. Matheus, and A. Wolisz. "Wireless Technology in Industrial Networks". In: *Proceedings of the IEEE* (June 2005). DOI: 10.1109/JPROC.2005.849717.

[137] Ian H. Witten, Radford M. Neal, and John G. Cleary. "Arithmetic Coding for Data Compression". In: *Commun. ACM* (June 1987). DOI: 10.1145/214762.214771.

[138] Yunnan Wu, Philip Chou, Kamal Jain, et al. "A Comparison of Network Coding and Tree Packing". In: *Information Theory, 2004. ISIT 2004. Proceedings. International Symposium On*. IEEE, 2004.

[139] Z. Yan, H. Xie, and B. W. Suter. "Rank Deficient Decoding of Linear Network Coding". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. May 2013. DOI: 10.1109/ICASSP.2013.6638629.

[140] Karen K. Yuen and W. J. Dixon. "The Approximate Behaviour and Performance of the Two-Sample Trimmed t". In: *Biometrika* (1973).

Author Publications

[87] B. Milic, S. Brack, and R. Naumann. "Hierarchical Configuration System for Wireless Mesh Networks in Manufacturing Industry". In: *Wireless and Mobile Networking Conference (WMNC), 2014 7th IFIP*. IEEE, 2014.

[89] R. Naumann, S. Dietzel, and B. Scheuermann. "INFLATE: Incremental Wireless Transmission for Sensor Information in Industrial Environments". In: *2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*. 2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS). Dec. 2015. DOI: 10.1109/ANTS.2015.7413631.

[90] R. Naumann, S. Dietzel, and B. Scheuermann. "Best of Both Worlds: Prioritizing Network Coding without Increased Space Complexity". In: *2016 IEEE 41st Conference on Local Computer Networks (LCN)*. 2016 IEEE 41st Conference on Local Computer Networks (LCN). Nov. 2016. DOI: 10.1109/LCN.2016.123.

[91] R. Naumann, S. Dietzel, and B. Scheuermann. "Towards More Realistic Network Simulations: Leveraging the System-Call Barrier". In: *Ad Hoc Networks*. Ed. by Yifeng Zhou and Thomas Kunz. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer International Publishing, 2017. DOI: 10.1007/978-3-319-51204-4_15.

[92] R. Naumann, S. Dietzel, and B. Scheuermann. "Push the Barrier: Discrete Event Protocol Emulation". In: *IEEE/ACM Transactions on Networking* (2019). DOI: 10.1109/TNET.2019.2897310.

[93] R. Naumann, S. Dietzel, L. Wartschinski, B. Schumacher, and B. Scheuermann. "TANDEM: Prioritizing Wireless Communication for Robust Industrial Process Control". In: *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. 2017 26th International Conference on Com-

puter Communication and Networks (ICCCN). July 2017. DOI: 10.1109/ICCCN.2017.8038458.

[94] R. Naumann, M. Schaeffer, S. Dietzel, and B. Scheuermann. "Prioritize Efficiently: Layer Selection in Network Coding". In: *Computer Communications* (2019). DOI: 10.1016/j.comcom.2019.05.008.

[114] M. Schaeffer, R. Naumann, S. Dietzel, and B. Scheuermann. "Hierarchical Layer Selection with Low Overhead in Prioritized Network Coding". In: *2018 IFIP Networking Conference (IFIP Networking)*. 2018 IFIP Networking Conference (IFIP Networking). 2018.

[115] M. Schaeffer, R. Naumann, S. Dietzel, and B. Scheuermann. "Poster: Impact of Prioritized Network Coding on Sensor Data Collection in Smart Factories". In: 2018 IFIP Networking Conference (IFIP Networking). 2018 IFIP Networking Conference (IFIP Networking), 2018.

[123] H. Sparka, R. Naumann, S. Dietzel, and B. Scheuermann. "Effective Lossless Compression of Sensor Information in Manufacturing Industry". In: *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*. 2017 IEEE 42nd Conference on Local Computer Networks (LCN). Oct. 2017. DOI: 10.1109/LCN.2017.89.